Db2 11 for z/OS

Application Programming and SQL Guide

SC19-4051-08

Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from PDF format manuals for Db2 11 for z/OS (Db2 for z/OS in IBM Documentation).

2022-02-01 edition

This edition applies to Db2® 11 for z/OS® (product number 5615-DB2), Db2 11 for z/OS Value Unit Edition (product number 5697-P43), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

Contents

About this information

This information discusses how to design and write application programs that access Db2 for z/OS (Db2), a highly flexible relational database management system (DBMS).

Throughout this information, "Db2" means "Db2 11 for z/OS". References to other Db2 products use complete names or specific abbreviations.

Important: To find the most up to date content for Db2 11 for z/OS, always use IBM® Documentation or download the latest PDF file from PDF format manuals for Db2 11 for z/OS (Db2 for z/OS in IBM Documentation).

This information assumes that Db2 11 is running in new-function mode, and that your application is running with the application compatibility value of 'V11R1', except for the following section that describe the migration process and how to activate new function:

- Migrating to Db2 11 (Db2 Installation and Migration)
- What's new in Db2 11 (Db2 for z/OS What's New?)
- Changes to plan for in Db2 11 (Db2 for z/OS What's New?)

Availability of new function in Db2 11

The behavior of data definition statements such as CREATE, ALTER, and DROP, which embed data manipulation SQL statements that contain new capabilities, depends on the application compatibility value that is in effect for the application. An application compatibility value of 'V11R1' must be in effect for applications to use new capability in embedded statements such as SELECT, INSERT, UPDATE, DELETE, MERGE, CALL, and SET *assignment-statement*. Otherwise, an application compatibility value of 'V10R1' can be used for data definition statements.

Generally, new SQL capabilities, including changes to existing language elements, functions, data manipulation statements, and limits, are available only in new-function mode with applications set to an application compatibility value of 'V11R1'.

Optimization and virtual storage enhancements are available in conversion mode unless stated otherwise.

SQL statements can continue to run with the same expected behavior as in DB2® 10 new-function mode with an application compatibility value of 'V10R1'.

Who should read this information

This information is for Db2 application developers who are familiar with Structured Query Language (SQL) and who know one or more programming languages that Db2 supports.

Db2 Utilities Suite for z/OS

Important: In Db2 11, the Db2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

Db2 11 utilities can use the DFSORT program regardless of whether you purchased a license for DFSORT on your system. For more information, see the following informational APARs:

- II14047
- II14213
- II13495

Db2 utilities can use IBM Db2 Sort for z/OS (5655-W42) as an alternative to DFSORT for utility SORT and MERGE functions. Use of Db2 Sort for z/OS requires the purchase of a Db2 Sort for z/OS license. For more information about Db2 Sort for z/OS, see Db2 Sort for z/OS.

Related concepts

Db2 utilities packaging (Db2 Utilities)

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

Db2
: Represents either the Db2 licensed program or a particular Db2 subsystem.

IBM re-branded DB2 to Db2, and Db2 for z/OS is the new name of the offering previously know as "DB2 for z/OS". For more information, see Revised naming for IBM Db2 family products on IBM z/OS platform. As a result, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. If the PID, Entitlement Entity, version, modification, and release information match, assume that they refer to the same product.

Tivoli® OMEGAMON® XE for Db2 Performance Expert on z/OS
: Refers to any of the following products:

- IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language
: Represent the C or C++ programming language.

CICS®
: Represents CICS Transaction Server for z/OS.

IMS
: Represents the IMS Database Manager or IMS Transaction Manager.

MVS™
: Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
: Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 11 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 11 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: IBM Documentation (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

How to send your comments about Db2 for z/OS documentation

Your feedback helps IBM to provide quality documentation.

End of support (EOS): Db2 11 reached EOS on March 31, 2021. The online product documentation is provided as-is for clients with extended service contracts. For more information, see End of support (March 31, 2021) (Db2 for z/OS in IBM Documentation).

Send any comments about Db2 for z/OS and related product documentation by email to db2zinfo@us.ibm.com.

To help us respond to your comment, include the following information in your email:

- The product name and version
- The address (URL) of the page, for comments about online documentation
- The book name and publication date, for comments about PDF manuals
- The topic or section title
- The specific text that you are commenting about and your comment

Related concepts
About this information (Db2 for z/OS in IBM Documentation)
Related reference
PDF format manuals for Db2 11 for z/OS (Db2 for z/OS in IBM Documentation)

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in Db2 for z/OS documentation:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►—— symbol indicates the beginning of a statement.

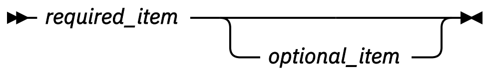
  The ——► symbol indicates that the statement syntax is continued on the next line.

  The ►—— symbol indicates that a statement is continued from the previous line.

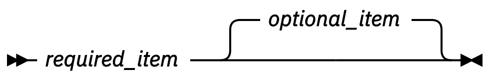
  The ——►◄ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).

  ►►— *required_item* —►◄

- Optional items appear below the main path.

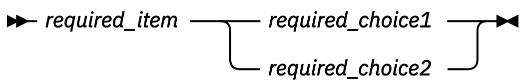


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

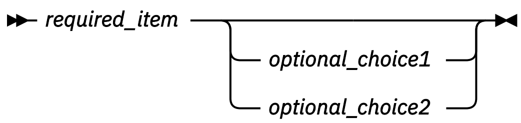


- If you can choose from two or more items, they appear vertically, in a stack.

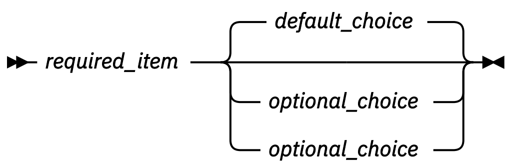
  If you *must* choose one of the items, one item of the stack appears on the main path.



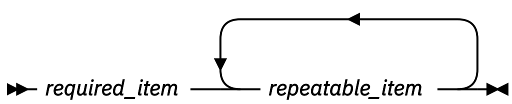
If choosing one of the items is optional, the entire stack appears below the main path.



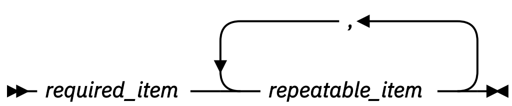
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

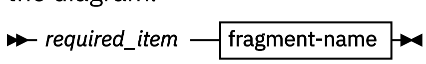


If the repeat arrow contains a comma, you must separate repeated items with a comma.

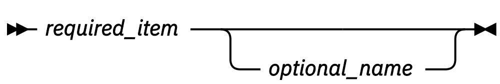


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



fragment-name



- For some references in syntax diagrams, you must follow any rules described in the description for that diagram, and also rules that are described in other syntax diagrams. For example:
  - For *expression*, you must also follow the rules described in Expressions (Db2 SQL).
  - For references to *fullselect*, you must also follow the rules described in fullselect (Db2 SQL).
  - For references to *search-condition*, you must also follow the rules described in Search conditions (Db2 SQL).
- With the exception of XPath keywords, keywords appear in uppercase (for example, `FROM`). Keywords must be spelled exactly as shown. XPath keywords are defined as lowercase names, and must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related concepts

Commands in Db2 (Db2 Commands)
Db2 online utilities (Db2 Utilities)
Db2 stand-alone utilities (Db2 Utilities)

Chapter 1. Planning for and designing Db2 applications

Before you write or run your program, you need to make some planning and design decisions. These decisions need to be made whether you are writing a new Db2 application or migrating an existing application from a previous release of Db2.

About this task

If you are migrating an existing application from a previous release of Db2, read the application and SQL release incompatibilities and make any necessary changes in the application.

If you are writing a new Db2 application, first determine the following items:

- the value of some of the SQL processing options
- the binding method
- the value of some of the bind options

Then make sure that your program implements the appropriate recommendations so that it promotes concurrency, can handle recovery and restart situations, and can efficiently access distributed data.

Related concepts
Tools and IDEs for developing Db2 applications (Introduction to Db2 for z/OS)
Related tasks
Programming applications for performance (Db2 Performance)
Programming for concurrency (Db2 Performance)
Writing efficient SQL queries (Db2 Performance)
Improving performance for applications that access distributed data (Db2 Performance)
Related reference
BIND and REBIND options for packages, plans, and services (Db2 Commands)

Application and SQL release incompatibilities

When you migrate from Db2 10 to Db2 11, be aware of and plan for application and SQL release incompatibilities that might affect your migration.

re

GUPI

Plan for the following changes in Db2 11 that might affect your migration.

Release incompatibilities that were changed or added since the first edition of this Db2 11 publication are indicated by a vertical bar in the left margin. In other areas of this publication, a vertical bar in the margin indicates a change or addition that has occurred since the Db2 10 release of this publication.

Change to determination of ASUTIME for dynamic statements

In Db2 11 new-function mode with application compatibility set to 'V11R1', the dynamic SQL ASUTIME limit for each routine is used by the resource limit facility.

Explanation

The ASUTIME limit that is specified for the routine determines the limit. If the dynamic SQL statements in a routine use more ASUTIME than the limit, then SQLCODE -905 is returned. This SQLCODE occurs even if the value is lower than the ASUTIME limit of a top-level calling package. The ASUTIME limit that is specified for the top-level calling package is not considered. In previous versions of Db2, SQLCODE -905 is issued when the limit of the top-level calling package is encountered. Now in Db2 11 the time that is used by each nested routine is compared to the ASUTIME limit.

Possible impact to your Db2 environment

Because the limit is enforced for each monitored routine, your applications might return more SQLCODE -905 errors.

Actions to take

While in conversion mode with application compatibility for your package set to value ’V10R1’, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier ’1103’. Review and, if necessary, adjust the ASUTIME limits on routines and packages that use dynamic SQL.

Related concepts

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

Related tasks

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Related information

-905 (Db2 Codes)

Automatic rebind of plans and packages created before DB2 9

Explanation

Plans and packages that were last bound before DB2 9 are not supported in Db2 11 conversion mode and later.

Possible impact to your Db2 environment

If you specify YES or COEXIST for the ABIND subsystem parameter, Db2 11 automatically rebinds plans and packages that were bound before DB2 9. As a result, an execution delay might occur the first time that such a plan or package is loaded. Also, the automatic rebind might change the access path to a potentially more efficient access path.

If you specify NO for the ABIND subsystem parameter, negative SQLCODEs are returned for each attempt to run a package or plan that was bound before DB2 9. SQLCODE -908, SQLSTATE 23510 is returned for packages, and SQLCODE -923, SQLSTATE 57015 is returned for plans until they are rebound in Db2 11.

Actions to take

To identify plans and packages that were bound before DB2 9, run the Db2 11 premigration job DSNTIJPM on your DB2 10 catalog.

Related reference

AUTO BIND field (ABIND subsystem parameter) (Db2 Installation and Migration)

Related information

-908 (Db2 Codes)

-923 (Db2 Codes)

Invalidated plans and packages

During the enabling-new-function mode processing, plans and packages that reference the affected Db2 catalog and directory table spaces become invalidated.

Explanation

The following table spaces in the Db2 catalog and directory are modified when you run job DSNTIJEN to enable Db2 11 new-function mode:

- DSNDB01.SYSUTILX
- DSNDB01.SYSLGRNX
- DSNDB06.SYSCOPY
- DSNDB06.SYSRTSTS
- DSNDB06.SYSTSIXS
- DSNDB06.SYSTSTAB
- DSNDB06.SYSSTR

As these table spaces are processed, Db2 invalidates packages or plans that reference them.

The packages that are dependent on the following catalog tables are also invalidated:

- SYSIBM.SYSCOPY
- SYSIBM.SYSCHECKS
- SYSIBM.SYSCHECKS2
- SYSIBM.SYSCHECKDEP
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSPENDINGDDL
- SYSIBM.SYSSTRINGS
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSINDEXSPACESTATS
- SYSIBM.SYSTABLESPACESTATS

Possible impact to your Db2 environment

If you have autobind enabled, the invalid packages are bound on the first run after they were marked invalid.

If you have autobind disabled, each attempt to use an invalidated package fails with SQLCODE -908 to indicate that the application must be bound before it can be run.

Actions to take

For SYSLGRNX, existing CHAR(6) columns were changed to CHAR(10). You might need to modify your application before it can run successfully.

For SYSUTILX, the RBA fields were moved to new fields. Applications might need to be updated before you can see the new fields.

The SYSCOPY table space was replaced by a new table space, SYSTSCPY. You might need to modify your application before it can run successfully.

The SYSRTSTS table space was replaced by two new table spaces, SYSTSTSS and SYSTSISS. SYSTSTSS contains the SYSIBM.SYSTABLESPACESTATS catalog table and SYSTSISS contains the

SYSIBM.SYSINDEXSPACESTATS table. You might need to modify your application before it can run successfully.

The SYSSTR table space was replaced by four new table spaces, SYSTSCKS, SYSTSCHX, SYSTSCKD, and SYSTSSRG. SYSTSCKS contains SYSIBM.SYSCHECKS, SYSTSCHX contains SYSIBM.SYSCHECKS2, SYSTSCKD contains SYSIBM.SYSCHECKDEP, and SYSTSSRG contains SYSIBM.SYSSTRINGS catalog table. You might need to modify your application before it can run successfully.

Default for ODBC limited block fetch

The default for the LIMITEDBLOCKFETCH initialization keyword changed.

Explanation

In DB2 10, ODBC limited block fetch was disabled by default. In Db2 11, ODBC limited block fetch is enabled by default.

Possible impact to your Db2 environment

Your applications might use limited block fetch, when they did not do so previously.

Actions to take

If the default is not appropriate for your ODBC applications, you can change it by modifying the value of the LIMITEDBLOCKFETCH initialization keyword.

Related reference

Db2 ODBC initialization keywords (Db2 Programming for ODBC)

Views, materialized query tables, and SQL table functions with period specifications

Explanation

In Db2 11, views, materialized query tables, and SQL table functions that were created with period specifications in DB2 10 are not supported.

Possible impact to your Db2 environment

If such views, materialized query tables, or SQL functions are used in Db2 11, incorrect results might occur.

Actions to take

To prepare for this change, drop all views, materialized query tables, and SQL table functions that contain a SYSTEM_TIME or BUSINESS_TIME period specification.

To identify such existing views, materialized query tables, and SQL table functions, run the Db2 11 premigration job DSNTIJPM on your DB2 10 catalog.

You can also manually issue the following queries.

To identify views and materialized query tables that were created with a period specification, issue the following query:

```
SELECT * FROM SYSIBM.SYSVIEWDEP WHERE BTYPE IN ('W', 'Z') AND DTYPE IN ('V', 'M');
```

To identify SQL table functions that were created with a period specification, issue the following query:

```
SELECT * FROM SYSIBM.SYSDEPENDENCIES WHERE BTYPE = 'Z';
```

To identify SQL scalar functions that were created with a period specification or period clause, issue the following query:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE BTYPE IN ('W', 'Z') AND DTYPE = 'N';
```

Related tasks

Run premigration queries (DSNTIJPM) (Db2 Installation and Migration)

Dropping columns named CLONE, ORGANIZATION, or VERSIONING

In Db2 11 new-function mode, a column that is named CLONE, ORGANIZATION, or VERSIONING should be specified as a delimited identifier in order to be dropped from a table.

Explanation

Prior to Db2 11, CLONE, ORGANIZATION, and VERSIONING are reserved keywords that can appear after the DROP keyword in an ALTER TABLE statement. When CLONE, ORGANIZATION, or VERSIONING is specified as a simple token (that is, not as a delimited identifier), these keywords can only match the DROP CLONE, DROP ORGANIZATION, or DROP VERSIONING clauses on an ALTER TABLE statement.

Possible impact to your Db2 environment

If you intend to drop a column named CLONE, ORGANIZATION, or VERSIONING in Db2 11, and the name is specified as a simple token on the ALTER TABLE statement, the Db2 subsystem might interpret the ALTER TABLE statement as specifying the DROP CLONE, DROP ORGANIZATION, or DROP VERSIONING clauses instead of the DROP COLUMN clause.

Actions to take

To drop a column named CLONE, ORGANIZATION, or VERSIONING in Db2 11, the name must be specified as a delimited identifier. For example: DROP "ORGANIZATION" or DROP "CLONE" (assuming " is the delimiter for a delimited identifier).

Alternatively, you can specify the optional COLUMN keyword in the DROP COLUMN clause. For example: DROP COLUMN ORGANIZATION or DROP COLUMN CLONE.

Related reference

ALTER TABLE (Db2 SQL)

Allow XPath processing to continue even if error on filtered results

In Db2 11 new-function mode with application compatibility set to ’V11R1’, XPath processing might return fewer errors on predicate expressions with an explicit cast or an operation with an invalid value.

Explanation

In previous versions, even though the invalid result is filtered from the result set, XPath processing would return an error SQLCODE. In Db2 11, examples of XPath expressions that have fewer errors include situations when:

- Data is filtered from the result by the predicate before an invalid operation such as division of a number by zero
- Data is explicitly cast to an incompatible data type

Possible impact to your Db2 environment

Your applications might return fewer error SQLCODEs.

Actions to take

While in conversion mode with application compatibility for your package set to value 'V10R1', run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier '1102'.

Related concepts

XML enhancements (Db2 for z/OS What's New?)

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

XML document node implicitly added on insert and update

In Db2 11 new-function mode with application compatibility set to 'V11R1', if an XML document does not have a document node, then one is added during insert and update operations.

Explanation

In previous versions of Db2, document nodes are not implicitly added and an SQL insert or update of an XML document returned SQLCODE -20345. To avoid the error, an application invokes the XMLDOCUMENT function before the insert or update. In Db2 11, an XML document node is added if one does not exist in the XML document.

Possible impact to your Db2 environment

Your applications might return fewer errors on insert and update operations.

Actions to take

While in conversion mode with application compatibility for your package set to value 'V10R1', run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier '1101'. In addition, you can review your applications for use of the XMLDOCUMENT function.

Related concepts

XML enhancements (Db2 for z/OS What's New?)

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

Client information special registers length

In Db2 11 new-function mode with application compatibility set to 'V11R1', special registers for client information fields might return different length values. The values in special registers CURRENT CLIENT_USERID, CURRENT CLIENT_WRKSTNAME, CURRENT CLIENT_APPLNAME, and CURRENT CLIENT_ACCTNG are determined by the application compatibility level.

Explanation

In previous versions of Db2, client information values were truncated and padded to the maximum length. In Db2 11 the following lengths increase:

- The maximum length of CURRENT CLIENT_USERID increases from 16 bytes to 128 bytes.
- The maximum length of CURRENT CLIENT_WKSTNAME increases from 18 bytes to 255 bytes.
- The maximum length of CURRENT CLIENT_APPLNAME increases from 32 bytes to 255 bytes.
- The maximum length of CURRENT CLIENT_ACCTNG increases from 200 bytes to 255 bytes.

In Db2 11, trailing blanks are removed.

Possible impact to your Db2 environment

When the application compatibility for your package is set to value 'V11R1', your applications might receive a different length client information value than they did previously. The value is no longer padded to the supported maximum length and trailing blanks are removed.

Actions to take

Review your applications for use of these special registers. While in conversion mode with application compatibility for your package set to value 'V10R1', run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier '1104', '1105', '1106', or '1107'.

Related concepts

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

Client information results from ADMIN_COMMAND_DB2

In Db2 11 conversion mode, the ADMIN_COMMAND_DB2 result set row returned changes in the created global temporary table SYSIBM.DB2_THREAD_STATUS when processing-type = "THD". The column data type and maximum lengths for WORKSTATION, USERID, APPLICATION, and ACCOUNTING change.

Explanation

In Db2 11 the following column data types and lengths change:

- WORKSTATION increases from CHAR(18) to VARCHAR(255).
- USERID increases from CHAR(16) to VARCHAR(128).
- APPLICATION increases from CHAR(32) to VARCHAR(255).
- ACCOUNTING increases from CHAR(247) to VARCHAR(255).

Possible impact to your Db2 environment

Your applications now receive a VARCHAR data type and possibly a different length client information value. The length is no longer padded to the supported maximum length.

Actions to take

Review your applications for use of the ADMIN_COMMAND_DB2 stored procedure.

Related reference

ADMIN_COMMAND_DB2 stored procedure (Db2 SQL)

ALTER statements that change limit keys are pending changes

Starting in Db2 11 new function mode, ALTER statements that modify limit key values result in pending data definition changes, which do not take effect until materialized by the REORG utility. They also block all subsequent immediate changes until materialized.

Explanation

In Db2 11, ALTER statements that change limit keys for the following types of table spaces result in pending data definition changes:

- partition-by-range spaces
- Partitioned (non-UTS) tables spaces with table-controlled partitioning

Affected partitions are placed in advisory REORG-pending (AREOR) status. In DB2 10, such ALTER statements resulted in immediate changes, which placed affected partitions in restrictive REORG-pending (REORP) status.

ALTER statements that alter the last partition are an exception if they change MAXVALUE to a value less than MAXVALUE for ascending, or from MINVALUE to a value greater than MINVALUE for descending. In such cases, the changes are immediate and affected partitions are placed in restricted REORG-pending status.

Possible impact to your Db2 environment

Affected partitions remain in advisory REORG-pending (AREOR) status, the old limit key values remain in effect, and the data remains available until the pending definition change is materialized. However, all subsequent immediate data definition changes, including in the same statement, remain blocked until the pending data definition changes are materialized. Table spaces with pending definition changes have an entry in SYSIBM.SYSPENDINGDDL.

Also, you can no longer materialize changes for ALTER statements that change limit keys by using the REORG TABLESPACE utility with SHRLEVEL NONE or the LOAD utility with REPLACE.

Actions to take

Modify any existing jobs that materialize limit key changes to run the REORG TABLESPACE utility with SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

For jobs that use the LOAD utility with REPLACE, modify the job to run the REORG TABLESPACE utility with SHRLEVEL CHANGE or SHRLEVEL REFERENCE before running LOAD.

Related concepts

Improved data availability when altering limit keys (Db2 for z/OS What's New?)
Pending data definition changes (Db2 Administration Guide)

Related tasks

Materializing pending definition changes (Db2 Administration Guide)
Altering table spaces (Db2 Administration Guide)

Related reference

REORG-pending status (Db2 Utilities)

SYSTABLEPART.LIMITKEY format variations

Explanation

Starting in Db2 11 conversion mode, the LIMITKEY column of the SYSTABLEPART catalog table can contain a mix of differently formatted values:

- Date and time values are delimited by single quotation marks (for example, `'2001-01-01'`). However, values that were added in releases prior to Db2 11 do not contain the delimiters.
- When the decimal point indicator is comma, a space follows any comma delimiter in the value. The comma decimal point indicator is used when the DECIMAL POINT IS field setting is `, (comma)` or a COBOL program that executes the ALTER statement uses the COMMA processing option. No space follows comma delimiters when period decimal point indicators are used, or for values added in releases prior to Db2 11.

Possible impact to your Db2 environment

Applications that do not tolerate the format variations might fail.

Actions to take

Modify any applications that use the LIMITKEY column to tolerate the format variations.

For example, to remove the single-quote delimiters for date and time values, from a single column in a partitioning key, you might use the following SQL statements:

```
SELECT DATE(STRIP(LIMITKEY,B,X'27'))  FROM SYSIBM.SYSTABLEPART WHERE …;
SELECT TIME(STRIP(LIMITKEY,B,X'27'))  FROM SYSIBM.SYSTABLEPART WHERE …;
```

If `(STRIP(LIMITKEY,B,X'27'))` is omitted from the statements, Db2 issues SQLCODE -180.

The use type of the column is also changed to S, which indicates a product-sensitive programming interface. It is expected that programs written to such interfaces might need to be changed in order to run with new product releases or versions, or as a result of service.

Related reference

SYSTABLEPART catalog table (Db2 SQL)

Programming interface information (Introduction to Db2 for z/OS)

DECIMAL POINT IS field (DECIMAL DECP value) (Db2 Installation and Migration)

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Removing the SYSPUBLIC schema from the SQL PATH routine option

Starting in Db2 11 conversion mode, SYSPUBLIC is the schema that is used for public aliases. As such, the SQL PATH routine option must not specify the SYSPUBLIC schema.

Explanation

In previous versions of Db2, you could not define functions, procedures, distinct types, and sequences in the SYSPUBLIC schema, but you were not restricted from specifying SYSPUBLIC as part of the SQL PATH. However, doing so no effect on applications. In Db2 11 you can no longer specify SYSPUBLIC as part of the SQL PATH.

Possible impact to your Db2 environment

Creation or resolution of some objects that worked in previous versions, might fail in Db2 11 with SQLCODE -713 if SYSPUBLIC is specified as part of the SQL PATH.

Actions to take

Query the catalog to see if any object schemas use SYSPUBLIC as the schema qualifier. This is highly unlikely for any object, but most likely with objects that use the SQL PATH (functions, procedures, distinct types, and sequences).

Change any existing SET PATH statements to not specify SYSPUBLIC as a schema.

Related concepts

SQL path (Db2 SQL)

Unqualified type, function, procedure, global variable, and specific names (Db2 SQL)

Related reference

CURRENT PATH (Db2 SQL)

SET PATH (Db2 SQL)

Related information

-713 (Db2 Codes)

SYSIBMADM schema added to the SQL path

In Db2 11 new-function mode with application compatibility set to ’V11R1’, SYSIBMADM is added to the SQL path as an implicit schema.

Explanation

If SYSIBMADM is not specified as an explicit schema in the SQL path, it is included as an implicit schema at the beginning of the path after SYSIBM, SYSFUN, and SYSPROC.

Possible impact to your Db2 environment

Applications that reference the content of the CURRENT PATH special register now have the SYSIBMADM schema returned when implicit schemas are included in the path. For example, the statement SELECT CURRENT PATH FROM SYSIBM.SYSDUMMY1 now returns "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","*authid*," where *authid* is the authorization ID of the statement, instead of "SYSIBM","SYSFUN","SYSPROC","*authid*."

Actions to take

No action is required.

Related concepts
SQL path (Db2 SQL)
Related reference
CURRENT PATH (Db2 SQL)

Change in result for CAST(*string* AS TIMESTAMP)

In Db2 11 new-function mode with application compatibility set to ’V11R1’, the result of CAST(*string* AS TIMESTAMP) is changed in some cases.

Explanation

Previously, when Db2 executed CAST(*string* AS TIMESTAMP), Db2 interpreted an 8-byte string as a Store Clock value and a 13-byte string as a GENERATE_UNIQUE value. This interpretation might result in an incorrect result from the CAST specification. Starting with Db2 11, with the application compatibility set to V11R1, when an 8-byte string or a 13-byte string is input to CAST(*string* AS TIMESTAMP), Db2 interprets the input strings as string representations of TIMESTAMP values.

Possible impact to your Db2 environment

An invalid representation of an 8-byte or 13-byte string in CAST(*string* AS TIMESTAMP) results in SQLCODE -180.

For example, suppose that you execute the following SQL statements in Db2 11 new-function mode:

```
-- SET APPLICATION COMPATIBILITY TO V10R1
SET CURRENT APPLICATION COMPATIBILITY='V10R1';
-- CAST AN 8-BYTE STRING REPRESENTATION OF A DATETIME VALUE
-- TO TIMESTAMP
SELECT CAST('1/1/2013' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
-- CAST AN 8-BYTE STRING REPRESENTATION OF A STORE CLOCK VALUE
-- TO TIMESTAMP
SELECT CAST(X'CAB5060708090100' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
```

The result is of the first SELECT statement is 2034-07-25-16.43.41.599503, which is an incorrect result. The result of the second SELECT statement is 2013-01-01-20.37.04.246928, which is the correct result if the input string is interpreted as a Store Clock value.

If you execute the following SQL statements, the results differ:

```
-- SET APPLICATION COMPATIBILITY TO V11R1
SET CURRENT APPLICATION COMPATIBILITY='V11R1';
-- CAST AN 8-BYTE STRING REPRESENTATION OF A DATETIME VALUE
-- TO TIMESTAMP
SELECT CAST('1/1/2013' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
-- CAST AN 8-BYTE STRING REPRESENTATION OF A STORE CLOCK VALUE
-- TO TIMESTAMP
SELECT CAST(X'CAB5060708090100' AS TIMESTAMP) FROM SYSIBM.SYSDUMMY1;
```

The result of the first SELECT statement is 2013-01-01-00.00.00.000000, which is the correct result. The result of the second SELECT statement is SQLCODE -180, because a Store Clock value is not valid input to CAST(*string* AS TIMESTAMP).

Actions to take

While in Db2 11 conversion mode, or in Db2 11 new-function mode with application compatibility set to V10R1, identify applications with this incompatibility by starting a trace for IFCID 0366 or IFCID 0376, and then running the applications. Review the trace output for incompatible changes with the identifier 1109. If you need to convert Store Clock values to the TIMESTAMP data type, use the TIMESTAMP built-in function instead of CAST(*string* AS TIMESTAMP).

For example:

```
-- SET APPLICATION COMPATIBILITY TO V11R1
SET CURRENT APPLICATION COMPATIBILITY='V11R1';
-- CONVERT AN 8-BYTE STRING REPRESENTATION OF A STORE CLOCK VALUE
-- TO TIMESTAMP
SELECT TIMESTAMP(X'CAB5060708090100') FROM SYSIBM.SYSDUMMY1;
```

You receive the correct result of 2013-01-01-20.37.04.246928.

Related reference

CAST specification (Db2 SQL)

New maximum lengths for values that are returned for some built-in functions

In Db2 11 new-function mode with application compatibility set to ’V11R1’, the maximum lengths for values that are returned for some built-in functions is decreased.

Explanation

For the SPACE and VARCHAR built-in functions, the maximum length of the result is changed from 32767 to 32764 bytes.

Possible impact to your Db2 environment

If the length of the output string for any of these functions is greater than 32764 bytes, SQLCODE -171 is returned.

Actions to take

Review your applications for use of these functions, and, if necessary, modify the function input so that the result does not exceed 32764 bytes. While in conversion mode with application compatibility for your package set to value ’V10R1’, run your applications with IFCID 0366 or IFCID 0376 enabled. Then, review the trace output for incompatible changes with the identifier ’1110’ or ’1111’.

Timestamp string representations

String representations of timestamp values must adhere to the rules in the SQL Reference. However, releases before DB2 10 inadvertently tolerate some string representations of timestamps with invalid syntax.

The behavior is controlled by the BIF_COMPATIBILITY subsystem parameter and the application compatibility setting.

- Db2 11 with application compatibility set to V11R1 strictly enforces valid string representations of timestamp values. This is equivalent to DB2 10 with the BIF_COMPATIBILITY subsystem parameter set to CURRENT.
- Db2 11 with application compatibility set to V10R1, the enforcement of valid string representations depends on the BIF_COMPATIBILITY value.

Actions to take

Review your setting of the BIF_COMPATIBILITY subsystem parameter. If the value is not CURRENT, the application compatibility for your package set to V10R1, and you have applications that require string representations of timestamp values supported in a pre-DB2 10, you should make appropriate changes to your SQL to use one of the supported formats.

To modify your applications:

1. Use IFCID 0366 or IFCID 0376 trace to identify applications that depend on the pre-DB2 10 formats.
2. Review the trace output with the function identifier '3' to identify SQL with unsupported timestamp values.
3. Make appropriate changes to your SQL statements.
4. Set the BIF_COMPATIBILITY value to CURRENT.

Related concepts

String representations of datetime values (Db2 SQL)

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

Data types of output arguments from a stored procedure call in a Java application

In Db2 11 new-function mode, when a Java™ application that uses the IBM Data Server Driver for JDBC and SQLJ calls a stored procedure, the data types of stored procedure output arguments match the data types of the parameters in the stored procedure definition.

Explanation

Before DB2 10, if a Java client called a Db2 for z/OS stored procedure, the data types of output arguments matched the data types of the corresponding CALL statement arguments. Starting in DB2 10, the data types of the output arguments match the data types of the parameters in the stored procedure definition.

In Db2 11 conversion mode, or when application compatibility is set to V10R1, you can set the DDF_COMPATIBILITY subsystem parameter to SP_PARMS_JV to keep the behavior that existed before DB2 10. However, when application compatibility is set to V11R1, SP_PARMS_JV is no longer supported.

In Db2 11 with application compatibility set to V11R1, if the version of the IBM Data Server Driver for JDBC and SQLJ is lower than 3.63 or 4.13, a `java.lang.ClassCastException` might be thrown when an output argument value is retrieved.

Actions to take

Take one of the following actions:

- Upgrade the IBM Data Server Driver for JDBC and SQLJ to version 3.63 or 4.13, or later.
- Modify the data types in `CallableStatement.registerOutParameter` method calls to match the parameter data types in the stored procedure definitions. You can set application compatibility to V10R1 and run a trace for IFCID 0366 or 0376 to identify affected applications. Trace records for those applications have a QW0366FN field value of 8.

Related concepts

Application compatibility levels in Db2
The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

CHAR9 and VARCHAR9 functions for compatibility with pre-DB2 10 string formatting of decimal data

DB2 10 changed the formatting of decimal data by the CHAR and VARCHAR built-in functions and CAST specifications with a CHAR or VARCHAR result type. In Db2 11 you can use alternative functions for compatibility with applications that require decimal to string output in the pre-DB2 10 formats:

- CHAR9 (Db2 SQL)
- VARCHAR9 (Db2 SQL)

Important: For portable applications that might run on platforms other than Db2 for z/OS, do not use the CHAR9 and VARCHAR9 functions. Other Db2 family products do not support the these functions.

Actions to take

Review your setting for the BIF_COMPATIBILITY subsystem parameter. If the value is not CURRENT, and you have applications that require decimal to string output in the pre-DB2 10 format, you can rewrite SQL statements to use the CHAR9 and VARCHAR9 functions instead. This approach enables the development of new applications that can accept the current string formatting of decimal data.

To modify your applications to take advantage of the CHAR9, VARCHAR9 functions:

1. Use an IFCID 0366 trace to identify applications that depend on the pre-DB2 10 formats.
2. Rewrite the SQL statements in the identified applications to use the CHAR9 and VARCHAR9 functions instead of the CHAR and VARCHAR functions.
3. Set the BIF_COMPATIBILITY value to CURRENT.

Related reference

BIF COMPATIBILITY field (BIF_COMPATIBILITY subsystem parameter) (Db2 Installation and Migration)

Subsystem parameter BIF_COMPATIBILITY and SQL schemas for compatibility with pre-DB2 10 string formatting of decimal data

DB2 10 changed the formatting of decimal data by the CHAR and VARCHAR built-in functions and CAST specifications with a CHAR or VARCHAR result type. You can temporarily override these changes on a subsystem level by setting the BIF_COMPATIBILITY subsystem parameter to one of the pre-DB2 10 settings. You can also temporarily override these changes on an application level by adding schema SYSCOMPAT_V9 to the front of the PATH bind option or CURRENT PATH special register. The latter approach works for CHAR and VARCHAR functions and does not affect CAST specifications.

The recommended approach is to modify your applications to handle the DB2 10 and later behavior for these functions, as described in the following steps.

Actions to take

To modify your applications to handle the DB2 10 and later behavior for CHAR, VARCHAR, and CAST:

1. Identify applications that need to be modified to handle this change. Run a trace for IFCID 0376 to identify affected applications.
2. Ensure that the BIF_COMPATIBILITY subsystem parameter is set to V9_DECIMAL_VARCHAR.

   To handle the change for the CHAR function only, you can set BIF_COMPATIBILITY to V9, and complete the following steps for the CHAR function.
3. Change any affected applications to handle the DB2 10 and later CHAR and VARCHAR behavior, including stored procedures, non-inline user-defined functions, and trigger packages. Rewrite affected CAST specifications with the appropriate CHAR or VARCHAR function and a CAST to the correct length if needed.
4. Rebind and prepare packages with the PATH(SYSCURRENT,SYSIBM) rebind option. Putting the SYSCURRENT schema at the beginning of the SQL path causes the DB2 10 and later behavior to be used for the CHAR and VARCHAR built-in functions.

   Repeat this step for native stored procedures (SQLPL) and non-inline SQL scalar functions.
5. For views that reference these casts or built-in functions, determine whether the view needs to be changed to have the expected output. Drop and re-create the views with the PATH(SYSCURRENT,SYSIBM) rebind option only if necessary. Rebind any applications that reference the views with the PATH(SYSCURRENT,SYSIBM) option to use the DB2 10 and later CHAR and VARCHAR built-in functions. Repeat this step for inline SQL scalar functions.
6. For materialized query tables or indexes on expressions that reference these casts or built-in functions, drop and re-create the materialized query tables or indexes on expressions with the PATH(SYSCURRENT,SYSIBM) rebind option. Issue the REFRESH TABLE statement for materialized query tables. Rebind any applications that reference the materialized query tables or indexes on expressions with the PATH(SYSCURRENT,SYSIBM) option to use the DB2 10 and later CHAR and VARCHAR built-in functions.
7. Change the value of the BIF_COMPATIBILITY subsystem parameter to CURRENT. When the subsystem parameter value is CURRENT, new applications, rebinds, and CREATE statements use the DB2 10 and later CHAR, VARCHAR, and CAST behavior.

Materialized query tables and expression-based indexes use the CHAR, VARCHAR, and CAST behavior that is specified during its creation. If a reference statement has a different behavior that is specified by the BIF_COMPATIBILITY parameter or a different path, the materialized query table or expression-based index is not used.

Related reference

BIF COMPATIBILITY field (BIF_COMPATIBILITY subsystem parameter) (Db2 Installation and Migration)

Change in CREATE TRIGGER statements with a WHEN clause in which the *search-condition* references a system-period temporal table

If a CREATE TRIGGER statement is issued before Db2 11 new-function mode, and the *triggered-action* contains a WHEN clause in which the *search-condition* references a system-period temporal table, the CREATE TRIGGER statement succeeds, and the trigger continues to work in Db2 11 new-function mode. However, when a CREATE TRIGGER statement with a WHEN clause in which the *search-condition* references a system-period temporal table is issued in Db2 11 new-function mode, the CREATE TRIGGER statement fails. This happens because in Db2 11, the trigger package is created with the SYSTIMESENSITIVE(YES) bind option, which does not allow a reference to the system-period temporal table.

Actions to take

To create a trigger that references a system-period temporal table in the *search-condition* of a WHEN clause, follow these steps:

4. Issue the REBIND TRIGGER PACKAGE command with option SYSTIMESENSITIVE(NO) on the trigger package that was generated when you performed step "2" on page 15.

1. Issue an ALTER TABLE statement with the DROP VERSIONING clause on the system-period temporal table to temporarily disconnect the system-period temporal table from the related history table.
2. Issue the CREATE TRIGGER statement.
3. Issue an ALTER TABLE statement with the ADD VERSIONING clause to redefine the system-period temporal table. In the ALTER TABLE statement, specify the related history table in the USE HISTORY TABLE clause.
4. Issue the REBIND TRIGGER PACKAGE command with option SYSTIMESENSITIVE(NO) on the trigger package that was generated when you performed step "2" on page 15.

Related reference

CREATE TRIGGER (Db2 SQL)

ALTER TABLE (Db2 SQL)

REBIND TRIGGER PACKAGE (DSN) (Db2 Commands)

SQL reserved words

PSPI

Db2 11 introduces several new SQL reserved words, which are listed in Reserved words (Db2 SQL).

In some cases, the use of these reserved words might cause an incompatibility in Db2 11 conversion mode, regardless of the setting of the APPLCOMPAT flag.

Actions to take

Collect IFCID 0366 trace records in Db2 10. Values 4, 5, and 6 for the QW0366FN field indicate instances of reserved words in applications that will cause an incompatibility in Db2 11. Adjust these applications by changing the reserved word to a delimited identifier or by using a word that is not reserved in Db2 11. PSPI

Built-in function and existing user-defined functions

For built-in and user-defined functions the combination of the function name and the parameter list form the *signature* that Db2 uses to identify the function. If the signatures of new or changed built-in functions in Db2 11 match the signatures existing user-defined functions, applications with unqualified references to the existing user-defined functions might start invoking the new or changed built-in functions instead of the user-defined functions. Db2 11 introduces the following built-in function changes:

GUPI

Table 1. New functions

Function name	Description
ARRAY_AGG (Db2 SQL)	The ARRAY_AGG function returns an array in which each value of the input set is assigned to an element of the array.
ARRAY_DELETE (Db2 SQL)	The ARRAY_DELETE function deletes elements from an array.
ARRAY_FIRST (Db2 SQL)	The ARRAY_FIRST function returns the minimum array index value of an array.
ARRAY_LAST (Db2 SQL)	The ARRAY_LAST function returns the maximum array index value of an array.
ARRAY_NEXT (Db2 SQL)	The ARRAY_NEXT function returns the next larger array index value, relative to a specified array index value.
ARRAY_PRIOR (Db2 SQL)	The ARRAY_PRIOR function returns the next smaller array index value, relative to a specified array index value.

Function name	Description
BLOCKING_THREADS (Db2 SQL)	The BLOCKING_THREADS function returns a table that contains one row for each lock or claim that threads hold against specified databases.
CARDINALITY (Db2 SQL)	The CARDINALITY function returns the number of elements in an array.
CHAR9 (Db2 SQL)	The CHAR9 function returns a fixed-length character string representation of the argument. The CHAR9 function is intended for compatibility with previous releases of Db2 for z/OS that depend on the result format that is returned for decimal input values in Version 9 and earlier. **Important:** For portable applications that might run on platforms other than Db2 for z/OS, use the CHAR function instead. Other Db2 family products do not support the CHAR9 function.
MAX_CARDINALITY (Db2 SQL)	The MAX_CARDINALITY function returns the maximum number of elements that an array can contain.
MEDIAN	The MEDIAN function returns the median of a set of numbers. This function can run only on an accelerator server.
TRIM_ARRAY (Db2 SQL)	The TRIM_ARRAY function deletes elements from the end of an ordinary array.
VARCHAR9 (Db2 SQL)	The VARCHAR9 function returns a fixed-length character string representation of the argument. The VARCHAR9 function is intended for compatibility with previous releases of Db2 for z/OS that depend on the result format that is returned for decimal input values in Version 9 and earlier. **Important:** For portable applications that might run on platforms other than Db2 for z/OS, use the VARCHAR function instead. Other Db2 family products do not support the VARCHAR9 function.

Table 1. New functions (continued)

GUPI

Actions to take

To continue to execute a user-defined function with the same name or signature as a new built-in function or signature, qualify the name of the existing user defined function in your application. For more information about Db2 resolves qualified and unqualified references to functions, see Function resolution (Db2 SQL).

SQLCODE changes

Some SQLCODE numbers and message text might have changed in Db2 11. Also, the conditions under which some SQLCODEs are issued might have changed. For more information, see New, changed, and deleted codes (Db2 Codes).

GUPI

Determining the value of any SQL processing options that affect the design of your program

When you process SQL statements in an application program, you can specify options that describe the basic characteristics of the program. You can also indicate how you want the output listings to look. Although most of these options do not affect how you design or code the program, a few options do.

About this task

SQL processing options specify program characteristics such as the following items:

- The host language in which the program is written
- The maximum precision of decimal numbers in the program
- How many lines are on a page of the precompiler listing

In many cases, you may want to accept the default value provided.

Procedure

Review the list of SQL processing options and decide the values for any options that affect the way that you write your program.

For example, you need to know if you are using NOFOR or STDSQL(YES) before you begin coding.

Related concepts

Db2 program preparation overview
Before you can run an application program on Db2 for z/OS, you need to prepare it. To prepare the program, create a load module, possibly one or more packages, and an application plan.

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Changes that invalidate packages

Changes to your program or database objects can invalidate packages.

A change to your program probably invalidates one or more of your packages. For some changes, you must bind a new object. For others, rebinding is sufficient. A package can also become invalid for reasons that do not depend on operations in your program. For example, when an index is dropped that is used in an access path by one of your queries, a package can become invalid.

Db2 might rebind invalid packages automatically the next time that the package is run. For details, see "Automatic rebinds" on page 879.

Db2 marks packages invalid when they depend on the target object, and sometimes on related objects that are affected by cascading effects, of the following actions:

- Altering tables:
  - Adding a TIME, TIMESTAMP, or DATE column when the default value for added rows is CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP (*p*) WITHOUT TIME ZONE, or CURRENT TIMESTAMP (*p*) WITH TIME ZONE respectively
  - Adding a constraint with a delete rule of SET NULL or CASCADE. Packages that depend on tables that cascade deletes to the altered parent table are also invalidated.
  - Adding a security label
  - Adding, changing, or rotating partitions in a partitioned (non-UTS) or partition-by range (UTS) table space

- Altering or dropping a column
- Renaming a column. Cascading effects apply. See "Cascading effects on packages of renaming a column" on page 19.
- When a column is renamed, cascading effects apply.
- Altering a column such that a view cannot regenerate
- Altering the AUDIT attribute
- Dropping a column. For pending definition changes, the package invalidation occurs when the pending definition change is applied to the table.
- Altering for hash organization, or dropping hash organization
- Adding or removing a BUSINESS_TIME period for temporal versioning
- Enabling or disabling transparent archiving
- Adding, altering, or dropping a materialized query table (MQT) definition
- Dropping a clone table
- Activating or deactivating row-level access control
- Activating column-level access control if the table has an enabled column, or deactivating column-level access control
- For created temporary tables, adding a column

• Altering views:
  - Altering a view to regenerate it

• Altering table spaces:
  - Changing the SBCS CCSID attribute
  - Increasing the MAXPARTITIONS attribute
  - Changing the SEGSIZE attribute to convert the table space to a partition-by-range (UTS) table space
  - Changing the DSSIZE attribute of a partitioned table space
  - Applying pending definition changes
  - Changing the buffer pool page size

• Altering indexes:
  - Adding a column
  - Altering an index to regenerate it
  - Altering the PADDED or NOT PADDED attribute
  - Altering a limit key value of a partitioning index
  - Specifying NOT CLUSTER for the partitioning index of a table that uses index-controlled partitioning, to convert the table to use table controlled partitioning
  - Applying pending definition changes

• Regenerating procedures. See the information about invalidation of packages in ALTER PROCEDURE (SQL - native) (Db2 SQL) for more information.

• Altering functions:
  - Altering an external function
  - Altering an inlined SQL scalar function
  - Altering a version of a compiled SQL scalar function to change certain options that are specified for the active version. See the information about invalidation of packages in ALTER FUNCTION (compiled SQL scalar) (Db2 SQL) for more information.
  - Altering SQL table functions:
    - Altering the SECURED or NOT SECTURED attribute

    - Altering the DETERMINISTIC or NOT DETERMINISTIC attribute, regardless of whether RESTRICT is specified
    - Regenerating a table function
- Enabling or disabling masks if column access control is in effect
- Dropping the package
- Dropping a package that provides the execute privilege for a plan
- Dropping objects such as aliases, global variables, indexes, materialized query tables, roles, sequences, tables, table spaces, triggers, views
- If column access control is enforced for a table, dropping row permissions or column masks
- Revoking authorization from the package owner to access a table, index, or view
- Revoking authorization from the package owner execute a stored procedure, if the package uses the CALL *procedure-name* form of the CALL statement to call the stored procedure
- Running the REORG utility with the REBALANCE keyword
- Running the REPAIR utility on a database with the DBD REBUILD option

Cascading effects on packages of renaming a column

ALTER TABLE RENAME COLUMN will invalidates any package that depends on the table in which the column is renamed. Any attempt to execute the invalidated package triggers an automatic rebind of the package.

The automatic rebind fails if the column is referenced in the package because the referenced column no longer exists in the table. In this case, applications that reference the package need to be modified, recompiled, and rebound to return the expected result.

The automatic rebind succeeds in either of the following cases:

- The package does not reference the column. In this case, the renaming of the column does not affect the query results that are returned by the package. The application does not need to be modified as a result of renaming the column.
- The package does reference the column, but after the column is renamed, another column with the name of the original column is added to the table. In this case, any query that references the name of the original column might return a different result set. In order to restore the expected results, the application would need to be modified to specify the new column name.

Example

The following scenario shows how renaming a column can cause a package to return unexpected results:

```
CREATE TABLE MYTABLE (MYCOL1 INT);
INSERT INTO TABLE MYTABLE
    VALUES (1);
SELECT MYCOL1 FROM MYTABLE -- this is the statement in
                                                     --  the package MYPACKAGE,
                           --  the query returns
                              --  a value of 1
ALTER TABLE MYTABLE
   RENAME COLUMN
    MYCOL1 TO MYCOL2;      -- MYPACKAGE is invalidated
                           --  and automatic rebind
                           --  of MYPACKAGE will fail
                           --  at this point
ALTER TABLE MYTABLE
   ADD COLUMN MYCOL1 VARCHAR(10);  -- automatic rebind
                                   --  of MYPACKAGE
                                   --  will be successful
INSERT INTO TABLE MYTABLE (MYCOL1)
   VALUES ('ABCD');
```

At this point an application executes MYPACKAGE, which results in a successful automatic rebind. However, the statement in the package will return `'ABCD'` instead of the expected `'1'`.

Related concepts

Automatic rebinds
Automatic rebinds (sometimes called "autobinds") occur when an authorized user runs a package or plan and the runtime structures in the plan or package cannot be used. This situation usually results from changes to the attributes of the data on which the package or plan depends, or changes to the environment in which the package or plan runs.

“Trigger packages” on page 161
A *trigger package* is a special type of package that is created only when you execute a CREATE TRIGGER statement. A trigger package executes only when the associated trigger is activated.

Invalidation of cached dynamic statements (Db2 Performance)

Related tasks

Identifying packages with characteristics that affect performance, concurrency, or the ability to run (Db2 Performance)

“Rebinding applications” on page 871
You must rebind applications to change bind options. You also need to rebind applications when you make changes that affect the plan or package, such as creating an index, but you have not changed the SQL statements.

Related reference

Invalid and inoperative packages (Managing Security)

Related information

00E30305 (Db2 Codes)

Identifying invalidated packages

You can identify packages that will become invalidated when certain changes are made to objects.

About this task

Certain changes to objects invalidate packages. By identifying these invalidated packages before you make the changes, you can prepare necessary rebind operations accordingly.

Procedure

To identify all packages that will be invalidated by a change to a specific object, run the following query:

```
SELECT   DISTINCT DCOLLID, DNAME, DTYPE
FROM     SYSIBM.SYSPACKDEP
WHERE    BQUALIFIER = object_qualifier
  AND    BNAME      = object_name
  AND    BTYPE      = object_type
ORDER BY DCOLLID, DNAME;
```

object_qualifier
The qualifier of the object

object_name
The name of the object

object_type
The type of object

Results

The query returns a table that contains package information based on the selected values in the query. For details about the selected values, see SYSPACKDEP catalog table (Db2 SQL).

Determining the value of any bind options that affect the design of your program

Several options of the BIND PACKAGE and BIND PLAN commands can affect your program design. For example, you can use a bind option to ensure that a package or plan can run only from a particular CICS connection or IMS region. Your code does not need to enforce this situation.

Procedure

Review the list of bind options and decide the values for any options that affect the way that you write your program.

For example, you should decide the values of the ACQUIRE and RELEASE options before you write your program. These options determine when your application acquires and releases locks on the objects it uses.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Programming applications for performance

You can achieve better Db2 performance by considering performance as you program and deploy your applications.

Procedure

To improve the performance of application programs that access data in Db2, use the following approaches when writing and preparing your programs:

- Program your applications for concurrency.

  The goal is to program and prepare applications in a way that:

  - Protects the integrity of the data that is being read or updated from being changed by other applications.
  - Minimizes the length of time that other access to the data is prevented.

  For more information about data concurrency in Db2 and recommendations for improving concurrency in your application programs, see the following topics:

  - Designing databases for concurrency (Db2 Performance)
  - Concurrency and locks (Db2 Performance)
  - Improving concurrency (Db2 Performance)
  - Improving concurrency in data sharing environments (Db2 Data Sharing Planning and Administration)
- Write SQL statements that access data efficiently.

  The predicates, subqueries, and other structures in SQL statements affect the access paths that Db2 uses to access the data.

  For information about how to write SQL statements that access data efficiently, see the following topics:

  - Ways to improve query performance (Introduction to Db2 for z/OS)
  - Writing efficient SQL queries (Db2 Performance)
- Use EXPLAIN or SQL optimization tools to analyze the access paths that Db2 chooses to process your SQL statements.

  By analyzing the access path that Db2 uses to access the data for an SQL statement, you can discover potential problems. You can use this information to modify your statement to perform better.

For information about how you can use EXPLAIN tables to analyze the access paths for your SQL statements, see the following topics:

- Investigating access path problems (Db2 Performance)
- 00C200A4 (Db2 Codes)
- Investigating SQL performance by using EXPLAIN (Db2 Performance)
- Interpreting data access by using EXPLAIN (Db2 Performance)
- EXPLAIN tables (Db2 Performance)
- EXPLAIN (Db2 SQL)

- Consider performance in the design of applications that access distributed data.

  The goal is to reduce the amount of network traffic that is required to access the distributed data, and to manage the use of system resources such as distributed database access threads and connections.

  For information about improving the performance of applications that access distributed data, see the following topics:

  - Ways to reduce network traffic (Introduction to Db2 for z/OS)
  - Managing Db2 threads (Db2 Performance)
  - Improving performance for applications that access distributed data (Db2 Performance)
  - Improving performance for SQL statements in distributed applications (Db2 Performance)

- Use stored procedures to improve performance, and consider performance when creating stored procedures.

  For information about stored procedures and Db2 performance, see the following topics:

  - Implementing Db2 stored procedures (Stored procedures provided by Db2)
  - Improving the performance of stored procedures and user-defined functions (Db2 Performance)

Related concepts

Query and application performance analysis (Introduction to Db2 for z/OS)

Programming for the instrumentation facility interface (IFI) (Db2 Performance)

Related tasks

Overview of programming applications that access Db2 for z/OS data

Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Planning for and designing Db2 applications

Before you write or run your program, you need to make some planning and design decisions. These decisions need to be made whether you are writing a new Db2 application or migrating an existing application from a previous release of Db2.

Designing your application for recovery

If your application fails or Db2 terminates abnormally, you need to ensure the integrity of any data that was manipulated in your application. You should consider possible recovery situations when you design your application.

Procedure

To design your application for recovery:

1. Put any changes that logically need to be made at the same time in the same unit of work. This action ensures that in case Db2 terminates abnormally or your application fails, the data is left in a consistent state.

   A *unit of work* is a logically distinct procedure that contains steps that change the data. If all the steps complete successfully, you want the data changes to become permanent. But, if any of the steps fail, you want all modified data to return to the original value before the procedure began. For example, suppose two employees in the sample table DSN8B10.EMP exchange offices. You need to exchange their office phone numbers in the PHONENO column. You need to use two UPDATE statements to make each phone number current. Both statements, taken together, are a unit of work. You want both statements to complete successfully. For example, if only one statement is successful, you want both phone numbers rolled back to their original values before attempting another update.

2. Consider how often you should commit any changes to the data.

   If your program abends or the system fails, Db2 backs out all uncommitted data changes. Changed data returns to its original condition without interfering with other system activities.

   For IMS and CICS applications, if the system fails, Db2 data does not always return to a consistent state immediately. Db2 does not process indoubt data (data that is neither uncommitted nor committed) until you restart IMS or the CICS attachment facility. To ensure that Db2 and IMS are synchronized, restart both Db2 and IMS. To ensure that Db2 and CICS are synchronized, restart both Db2 and the CICS attachment facility.

3. Consider whether your application should intercept abends.

   If your application intercepts abends, Db2 commits work, because it is unaware that an abend has occurred. If you want Db2 to roll back work automatically when an abend occurs in your program, do not let the program or run time environment intercept the abend. If your program uses Language Environment®, and you want Db2 to roll back work automatically when an abend occurs in the program, specify the run time options ABTERMENC(ABEND) and TRAP(ON).

4. **For TSO applications only:** Issue COMMIT statements before you connect to another DBMS.

   If the system fails at this point, Db2 cannot know whether your transaction is complete. In this case, as in the case of a failure during a one-phase commit operation for a single subsystem, you must make your own provision for maintaining data integrity.

5. **For TSO applications only:** Determine if you want to provide an abend exit routine in your program.

   If you provide this routine, it must use tracking indicators to determine if an abend occurs during Db2 processing. If an abend does occur when Db2 has control, you must allow task termination to complete. Db2 detects task termination and terminates the thread with the ABRT parameter. Do not re-run the program.

   Allowing task termination to complete is the only action that you can take for abends that are caused by the CANCEL command or by DETACH. You cannot use additional SQL statements at this point. If you attempt to execute another SQL statement from the application program or its recovery routine, unexpected errors can occur.

Related concepts

Unit of work (Introduction to Db2 for z/OS)

Unit of work in TSO

Applications that use the TSO attachment facility can explicitly define units of work by using the SQL COMMIT and ROLLBACK statements.

In TSO applications, a unit of work starts when the first updates of a Db2 object occur. A unit of work ends when one of the following conditions occurs:

- The program issues a subsequent COMMIT statement. At this point in the processing, your program has determined that the data is consistent; all data changes that were made since the previous commit point were made correctly.
- The program issues a subsequent ROLLBACK statement. At this point in the processing, your program has determined that the data changes were not made correctly and, therefore, should not be permanent. A ROLLBACK statement causes any data changes that were made since the last commit point to be backed out.
- The program terminates and returns to the DSN command processor, which returns to the TSO Terminal Monitor Program (TMP).

The first and third conditions in the preceding list are called a commit point. A *commit point* occurs when you issue a COMMIT statement or your program terminates normally.

Related reference

COMMIT (Db2 SQL)

ROLLBACK (Db2 SQL)

Unit of work in CICS

CICS applications can explicitly define units of work by using the CICS SYNCPOINT command. Alternatively, units of work are defined implicitly by several logic-breaking points.

All the processing that occurs in your program between two commit points is known as a logical unit of work (LUW) or unit of work. In CICS applications, a unit of work is marked as complete by a commit or synchronization (sync) point, which is defined in one of following ways:

- Implicitly at the end of a transaction, which is signaled by a CICS RETURN command at the highest logical level.
- Explicitly by CICS SYNCPOINT commands that the program issues at logically appropriate points in the transaction.
- Implicitly through a DL/I PSB termination (TERM) call or command.
- Implicitly when a batch DL/I program issues a DL/I checkpoint call. This call can occur when the batch DL/I program shares a database with CICS applications through the database sharing facility.

For example, consider a program that subtracts the quantity of items sold from an inventory file and then adds that quantity to a reorder file. When both transactions complete (and not before) and the data in the two files is consistent, the program can then issue a DL/I TERM call or a SYNCPOINT command. If one of the steps fails, you want the data to return to the value it had before the unit of work began. That is, you want it rolled back to a previous point of consistency. You can achieve this state by using the SYNCPOINT command with the ROLLBACK option.

By using a SYNCPOINT command with the ROLLBACK option, you can back out uncommitted data changes. For example, a program that updates a set of related rows sometimes encounters an error after updating several of them. The program can use the SYNCPOINT command with the ROLLBACK option to undo all of the updates without giving up control.

The SQL COMMIT and ROLLBACK statements are not valid in a CICS environment. You can coordinate Db2 with CICS functions that are used in programs, so that Db2 and non-Db2 data are consistent.

Planning for program recovery in IMS programs

To be prepared for recovery situations for IMS programs that access Db2 data, you need to make several design decisions that are specific to IMS programs. These decisions are in addition to the general recommendations that you should follow when designing your application for recovery.

About this task

Both IMS and Db2 handle recovery in an IMS application program that accesses Db2 data. IMS coordinates the process, and Db2 handles recovery for Db2 data.

Procedure

To plan for program recovery in IMS programs:

1. For a program that processes messages as its input, decide whether to specify single-mode or multiple-mode transactions on the TRANSACT statement of the APPLCTN macro for the program.

   Single-mode

   Indicates that a commit point in Db2 occurs each time the program issues a call to retrieve a new message. Specifying single-mode can simplify recovery; if the program abends, you can restart the program from the most recent call for a new message. When IMS restarts the program, the program starts by processing the next message.

Multiple-mode
Indicates that a commit point occurs when the program issues a checkpoint call or when it terminates normally. Those two events are the only times during the program that IMS sends the program's output messages to their destinations. Because fewer commit points are processed in multiple-mode programs than in single-mode programs, multiple-mode programs could perform slightly better than single-mode programs. When a multiple-mode program abends, IMS can restart it only from a checkpoint call. Instead of having only the most recent message to reprocess, a program might have several messages to reprocess. The number of messages to process depends on when the program issued the last checkpoint call.

Db2 does some processing with single- and multiple-mode programs. When a multiple-mode program issues a call to retrieve a new message, Db2 performs an authorization check and closes all open cursors in the program.

2. Decide whether to issue checkpoint calls (CHKP) and if so, how often to issue them.

   Each call indicates to IMS that the program has reached a sync point and establishes a place in the program from which you can restart the program.

   Consider the following factors when deciding when to use checkpoint calls:

   - How long it takes to back out and recover that unit of work. The program must issue checkpoints frequently enough to make the program easy to back out and recover.
   - How long database resources are locked in Db2 and IMS.
   - For multiple-mode programs: How you want the output messages grouped. Checkpoint calls establish how a multiple-mode program groups its output messages. Programs must issue checkpoints frequently enough to avoid building up too many output messages.

   Restriction: You cannot use SQL COMMIT and ROLLBACK statements in the Db2 DL/I batch support environment, because IMS coordinates the unit of work.

3. Issue CLOSE CURSOR statements before any checkpoint calls or GU calls to the message queue, not after.
4. After any checkpoint calls, set the value of any special registers that were reset if their values are needed after the checkpoint:

   A CHKP call causes IMS to sign on to Db2 again, which resets the special registers that are shown in the following table.

Table 2. Special registers that are reset by a checkpoint call.

Special register	Value to which it is reset after a checkpoint call
CURRENT PACKAGESET	blanks
CURRENT SERVER	blanks
CURRENT SQLID	blanks
CURRENT DEGREE	1

5. After any commit points, reopen the cursors that you want and re-establish positioning
6. Decide whether to specify the WITH HOLD option for any cursors.

   This option determines whether the program retains the position of the cursor in the Db2 database after you issue IMS CHKP calls. You always lose the program database positioning in DL/I after an IMS CHKP call.

   The program database positioning in Db2 is affected according to the following criteria:

   - If you do not specify the WITH HOLD option for a cursor, you lose the position of that cursor.
   - If you specify the WITH HOLD option for a cursor and the application is message-driven, you lose the position of that cursor.

- If you specify the WITH HOLD option for a cursor and the application is operating in DL/I batch or DL/I BMP, you retain the position of that cursor.

7. Use IMS rollback calls, ROLL and ROLB, to back out Db2 and DL/I changes to the last commit point.

   These options have the following differences:

   ROLL

   Specifies that all changes since the last commit point are to be backed out and the program is to be terminated. IMS terminates the program with user abend code U0778 and without a storage dump.

   When you issue a ROLL call, the only option you supply is the call function, ROLL.

   ROLLB

   Specifies that all changes since the last commit point are to be backed out and control is to be returned to the program so that it can continue processing.

   A ROLB call has the following options:

   - The call function, ROLB
   - The name of the I/O PCB

   How ROLL and ROLB calls effect DL/I changes in a batch environment depends on the IMS system log and back out options that are specified, as shown in the following table.

Table 3. Effects of ROLL and ROLLB calls on DL/I changes in a batch environment

	Options specified		
Rollback call	**System log option**	**Backout option**	**Result**
ROLL	tape	any	DL/I does not back out updates, and abend U0778 occurs. Db2 backs out updates to the previous checkpoint.
	disk	BKO=NO	
	disk	BKO=YES	DL/I backs out updates, and abend U0778 occurs. Db2 backs out updates to the previous checkpoint.

Table 3. Effects of ROLL and ROLLB calls on DL/I changes in a batch environment (continued)

	Options specified		
Rollback call	**System log option**	**Backout option**	**Result**
ROLB	tape	any	DL/I does not back out updates, and an AL status code is returned in the PCB. Db2 backs out updates to the previous checkpoint. The Db2 DL/I support causes the application program to abend when ROLB fails.
	disk	BKO=NO	
	disk	BKO=YES	DL/I backs out database updates, and control is passed back to the application program. Db2 backs out updates to the previous checkpoint. **Restriction:** You cannot specify the address of an I/O area as one of the options on the call; if you do, your program receives an AD status code. However, you must have an I/O PCB for your program. Specify CMPAT=YES on the CMPAT keyword in the PSBGEN statement for your program's PSB.

Related concepts

Checkpoints in IMS programs
Issuing checkpoint calls releases locked resources and establishes a place in the program from which you can restart the program. The decision about whether your program should issue checkpoints (and if so, how often) depends on your program.

Unit of work in IMS online programs

IMS applications can explicitly define units of work by using a CHKP, SYNC, ROLL, or ROLB call, or, for single-mode transactions, a GU call.

In IMS, a unit of work starts when one of the following events occurs:

- When the program starts
- After a CHKP, SYNC, ROLL, or ROLB call has completed
- For single-mode transactions, when a GU call is issued to the I/O PCB

A unit of work ends when one of the following events occurs:

- The program issues either a subsequent CHKP or SYNC call, or, for single-mode transactions, a GU call to the I/O PCB. At this point in the processing, the data is consistent. All data changes that were made since the previous commit point are made correctly.

- The program issues a subsequent ROLB or ROLL call. At this point in the processing, your program has determined that the data changes are not correct and, therefore, that the data changes should not become permanent.
- The program terminates.

Restriction: The SQL COMMIT and ROLLBACK statements are not valid in an IMS environment.

A commit point occurs in a program as the result of any one of the following events:

- The program terminates normally. Normal program termination is always a commit point.
- The program issues a checkpoint call. *Checkpoint calls* are a program's means of explicitly indicating to IMS that it has reached a commit point in its processing.
- The program issues a SYNC call. A *SYNC call* is a Fast Path system service call to request commit-point processing. You can use a SYNC call only in a non-message-driven Fast Path program.
- For a program that processes messages as its input, a commit point can occur when the program retrieves a new message. This behavior depends on the mode that you specify in the APPLCTN macro for the program:
  - If you specify single-mode transactions, a commit point in Db2 occurs each time the program issues a call to retrieve a new message.
  - If you specify multiple-mode transactions or you do not specify a mode, a commit point occurs when the program issues a checkpoint call or when it terminates normally.

At the time of a commit point, the following actions occur:

- IMS and Db2 can release locks that the program has held since the last commit point. Releasing these locks makes the data available to other application programs and users.
- Db2 closes any open cursors that the program has been using.
- IMS and Db2 make the program's changes to the database permanent.
- If the program processes messages, IMS sends the output messages that the application program produces to their final destinations. Until the program reaches a commit point, IMS holds the program's output messages at a temporary destination.

If the program abends before reaching the commit point, the following actions occur:

- Both IMS and Db2 back out all the changes the program has made to the database since the last commit point.
- IMS deletes any output messages that the program has produced since the last commit point (for nonexpress PCBs).
- If the program processes messages, people at terminals and other application programs receive information from the terminating application program.

If the system fails, a unit of work resolves automatically when Db2 and IMS batch programs reconnect. Any indoubt units of work are resolved at reconnect time.

Specifying checkpoint frequency in IMS programs

A checkpoint indicates a commit point in IMS programs. You should specify checkpoint frequency in your program in a way that allows it to easily be changed, in case the frequency that you initially specify is not appropriate.

Procedure

To specify checkpoint frequency in IMS programs:

1. Use a counter in your program to keep track of one of the following items:
   - Elapsed time
   - The number of root segments that your program accesses
   - The number of updates that your program performs

2. Issue a checkpoint call after a certain time interval, number of root segments, or number of updates.

Checkpoints in IMS programs

Issuing checkpoint calls releases locked resources and establishes a place in the program from which you can restart the program. The decision about whether your program should issue checkpoints (and if so, how often) depends on your program.

Generally, the following types of programs should issue checkpoint calls:

- Multiple-mode programs
- Batch-oriented BMPs
- Nonmessage-driven Fast Path programs. (These programs can use a special Fast Path call, but they can also use symbolic checkpoint calls.)
- Most batch programs
- Programs that run in a data sharing environment. (Data sharing makes it possible for online and batch application programs in separate IMS systems, in the same or separate processors, to access databases concurrently. Issuing checkpoint calls frequently in programs that run in a data sharing environment is important, because programs in several IMS systems access the database.)

You do not need to issue checkpoints in the following types of programs:

- Single-mode programs
- Database load programs
- Programs that access the database in read-only mode (defined with the processing option GO during a PSBGEN) and are short enough to restart from the beginning
- Programs that, by their nature, must have exclusive use of the database

A CHKP call causes IMS to perform the following actions:

- Inform Db2 that the changes that your program made to the database can become permanent. Db2 makes the changes to Db2 data permanent, and IMS makes the changes to IMS data permanent.
- Send a message that contains the checkpoint identification that is given in the call to the system console operator and to the IMS master terminal operator (MTO).
- Return the next input message to the program's I/O area if the program processes input messages. In MPPs and transaction-oriented BMPs, a checkpoint call acts like a call for a new message.
- Sign on to Db2 again.

Programs that issue symbolic checkpoint calls can specify as many as seven data areas in the program that is to be restored at restart. Db2 always recovers to the last checkpoint. You must restart the program from that point.

If you use symbolic checkpoint calls, you can use a restart call (XRST) to restart a program after an abend. This call restores the program's data areas to the way they were when the program terminated abnormally, and it restarts the program from the last checkpoint call that the program issued before terminating abnormally.

Restriction: For BMP programs that process Db2 databases, you can restart the program only from the latest checkpoint and not from any checkpoint, as in IMS.

Checkpoints in MPPs and transaction-oriented BMPs

In single-mode programs, checkpoint calls and message retrieval calls (called get-unique calls) both establish commit points. The checkpoint calls retrieve input messages and take the place of get-unique calls. BMPs that access non-DL/I databases and MPPs can issue both get unique calls and checkpoint calls to establish commit points. However, message-driven BMPs must issue checkpoint calls rather than get-unique calls to establish commit points, because they can restart from a checkpoint only. If a program abends after issuing a get-unique call, IMS backs out the database updates to the most recent commit point, which is the get-unique call.

In multiple-mode BMPs and MPPs, the only commit points are the checkpoint calls that the program issues and normal program termination. If the program abends and it has not issued checkpoint calls, IMS backs out the program's database updates and cancels the messages that it has created since the beginning of the program. If the program has issued checkpoint calls, IMS backs out the program's changes and cancels the output messages it has created since the most recent checkpoint call.

Checkpoints in batch-oriented BMPs

If a batch-oriented BMP does not issue checkpoints frequently enough, IMS can abend that BMP or another application program for one of the following reasons:

- Other programs cannot get to the data that they need within a specified amount of time.

  If a BMP retrieves and updates many database records between checkpoint calls, it can monopolize large portions of the databases and cause long waits for other programs that need those segments. (The exception to this situation is a BMP with a processing option of GO; IMS does not enqueue segments for programs with this processing option.) Issuing checkpoint calls releases the segments that the BMP has enqueued and makes them available to other programs.
- Not enough storage is available for the segments that the program has read and updated.

  If IMS is using program isolation enqueuing, the space that is needed to enqueue information about the segments that the program has read and updated must not exceed the amount of storage that is defined for the IMS system. (The amount of storage available is specified during IMS system definition.) If a BMP enqueues too many segments, the amount of storage that is needed for the enqueued segments can exceed the amount of available storage. In that case, IMS terminates the program abnormally. You then need to increase the program's checkpoint frequency before rerunning the program.

When you issue a DL/I CHKP call from an application program that uses Db2 databases, IMS processes the CHKP call for all DL/I databases, and Db2 commits all the Db2 database resources. No checkpoint information is recorded for Db2 databases in the IMS log or the Db2 log. The application program must record relevant information about Db2 databases for a checkpoint, if necessary. One way to record such information is to put it in a data area that is included in the DL/I CHKP call.

Performance might be slowed by the commit processing that Db2 does during a DL/I CHKP call, because the program needs to re-establish position within a Db2 database. The fastest way to re-establish a position in a Db2 database is to use an index on the target table, with a key that matches one-to-one with every column in the SQL predicate.

Recovering data in IMS programs

Online IMS systems handle recovery and restart. For a batch region, the operational procedures control recovery and restart for your location.

Procedure

Take one or more of the following actions depending on the type of program:

Program type	Recommended action
DL/I batch applications	Use the DL/I batch backout utility to back out DL/I changes. Db2 automatically backs out changes whenever the application program abends.
Applications that use symbolic checkpoints	Use a restart call (XRST) to restart a program after an abend. This call restores the program's data areas to the way they were when the program terminated abnormally, and it restarts the program from the last checkpoint call that the program issued before terminating abnormally.
BMP programs that access Db2 databases	Restart the program from the latest checkpoint.

Program type	Recommended action
	Restriction: You can restart the program only from the latest checkpoint and not from any checkpoint, as in IMS.
Applications that use online IMS systems	No action needed. Recovery and restart are part of the IMS system
Applications that reside in the batch region	Follow your location's operational procedures to control recovery and restart.

Undoing selected changes within a unit of work by using savepoints

Savepoints enable you to undo selected changes within a unit of work. Your application can set any number of savepoints and then specify a specific savepoint to indicate which changes to undo within the unit of work.

Procedure

To undo selected changes within a unit of work by using savepoints:

1. Set any savepoints by using SQL SAVEPOINT statements.

   Savepoints set a point to which you can undo changes within a unit of work.

   Consider the following abilities and restrictions when setting savepoints:

   - You can set a savepoint with the same name multiple times within a unit of work. Each time that you set the savepoint, the new value of the savepoint replaces the old value.
   - If you do not want a savepoint to have different values within a unit of work, use the UNIQUE option in the SAVEPOINT statement. If an application executes a SAVEPOINT statement with the same name as a savepoint that was previously defined as unique, an SQL error occurs.
   - If you set a savepoint before you execute a CONNECT statement, the scope of that savepoint is the local site. If you set a savepoint after you execute the CONNECT statement, the scope of that savepoint is the site to which you are connected.
   - When savepoints are active, which they are until the unit of work completes, you cannot access remote sites by using three-part names or aliases for three-part names. You can, however, use DRDA access with explicit CONNECT statements.
   - You cannot use savepoints in global transactions, triggers, user-defined functions, or stored procedures that are nested within triggers or user-defined functions.
2. Specify the changes that you want to undo within a unit of work by using the SQL ROLLBACK TO SAVEPOINT statement.

   Db2 undoes all changes since the specified savepoint. If you do not specify a savepoint name, Db2 rolls back work to the most recently created savepoint.
3. Optional: If you no longer need a savepoint, delete it by using the SQL RELEASE SAVEPOINT statement.

   Recommendation: If you no longer need a savepoint before the end of a transaction, release it. Otherwise, savepoints are automatically released at the end of a unit of work. Releasing savepoints is essential if you need to use three-part names to access remote locations, because you cannot perform this action while savepoints are active.

Examples

Example: Rolling back to the most recently created savepoint

When the ROLLBACK TO SAVEPOINT statement is executed in the following code, Db2 rolls back work to savepoint B.

```
EXEC SQL SAVEPOINT A;
...
```

```
EXEC SQL SAVEPOINT B;
...
EXEC SQL ROLLBACK TO SAVEPOINT;
```

Example: Setting savepoints during distributed processing

An application performs the following tasks:

1. Sets savepoint C1.
2. Does some local processing.
3. Executes a CONNECT statement to connect to a remote site.
4. Sets savepoint C2.

Because savepoint C1 is set before the application connects to a remote site, savepoint C1 is known only at the local site. However, because savepoint C2 is set after the application connects to the remote site, savepoint C2 is known only at the remote site.

Setting multiple savepoints with the same name

Suppose that the following actions occur within a unit of work:

1. Application A sets savepoint S.
2. Application A calls stored procedure P.
3. Stored procedure P sets savepoint S.
4. Stored procedure P executes the following statement: `ROLLBACK TO SAVEPOINT S`

When Db2 executes the ROLLBACK statement, Db2 rolls back work to the savepoint that was set in the stored procedure, because that value is the most recent value of savepoint S.

Related reference

RELEASE SAVEPOINT (Db2 SQL)
ROLLBACK (Db2 SQL)
SAVEPOINT (Db2 SQL)

Planning for recovery of table spaces that are not logged

To suppress logging, you can specify the NOT LOGGED option when you create or alter a table space. However, because logs are generally used in recovery, planning for recovery of table spaces for which changes are not logged requires some additional planning.

About this task

Although you can plan for recovery, you still need to take some corrective actions after any system failures to recover the data and fix any affected table spaces. For example, if a table space that is not logged was open for update at the time that Db2 terminates, the subsequent restart places that table space in LPL and marks it with RECOVER-pending status. You need to take corrective action to clear the RECOVER-pending status.

Procedure

To plan for recovery of table spaces that are not logged:

1. Ensure that you can recover lost data by performing one of the following actions:
   - Ensure that you have a data recovery source that does not rely on a log record to re-create any lost data.
   - Limit modifications that are not logged to easily repeatable changes that can be quickly repeated.
2. Avoid placing a table space that is not logged in a RECOVER-pending status.

   The following actions place a table space in RECOVER-pending status:

   - Issuing a ROLLBACK statement or ROLLBACK TO SAVEPOINT statement after modifying a table in a table space that is not logged.

- Causing duplicate keys or referential integrity violations when you modify a table space that is not logged.

  If the table space is placed in RECOVER-pending status, it is unavailable until you manually fix it.

3. For table spaces that are not logged and have associated LOB or XML table spaces, take image copies as a recovery set.

   This action ensures that the base table space and all the associated LOB or XML table spaces are copied at the same point in time. A subsequent RECOVER TO LASTCOPY operation for the entire set results in consistent data across the base table space and all of the associated LOB and XML table spaces.

Related tasks

Clearing the RECOVER-pending status (Db2 Administration Guide)

Related reference

RECOVER (Db2 Utilities)

Designing your application to access distributed data

You can design applications that access data on another database management system (DBMS) other than your local system. You should consider the limitations and recommendations for such programs when designing them.

Procedure

To design your application to access distributed data:

1. Ensure that the appropriate authorization ID has been granted authorization at the remote server to connect to that server and use resources from it.
2. If your application contains SQL statements that run at the requester, include at the requester a database request module (DBRM) that is bound directly into a package that is included in the plan's package list.
3. Copy the requester package to any remote server that is accessed by the application via a bind package copy command and include the remote packages in the application plan's package list.

   Recommendation: Specify an asterisk (*) instead of a specific name in the location name of any package entry of a plan so that the plan does not have to be rebound whenever a new location is accessed by the application or a different location is to be accessed.

4. For TSO and batch applications that update data at a remote server, ensure that one of the following conditions is true:

   - No other connections exist.
   - All existing connections are to servers that are restricted to read-only operations.

   Restriction: If neither of these conditions are met, the application is restricted to read-only operations.

   If one of these conditions is met, and if the first connection in a logical unit of work is to a server that supports two-phase commit, that server and all servers that support two-phase commit can update data. However, if the first connection is to a server that does not support two-phase commit, only that server is allowed to update data.

5. For programs that access at least one restricted system, ensure that your program does not violate any of the limitations for accessing restricted systems.

   A *restricted system* is a DBMS that does not implement two-phase commit processing.

   Accessing restricted systems has the following limitations:

   - For programs that access CICS or IMS, you cannot update data on restricted systems.
   - Within a unit of work, you cannot update a restricted system after updating a non-restricted system.
   - Within a unit of work, if you update a restricted system, you cannot update any other systems.

If you are accessing a mixture of systems, some of which might be restricted, you can perform the following actions:

- Read from any of the systems at any time.
- Update any one system many times in one unit of work.
- Update many systems, including CICS or IMS, in one unit of work, provided that none of them is a restricted system. If the first system you update in a unit of work is not restricted, any attempt to update a restricted system in that unit of work returns an error.
- Update one restricted system in a unit of work, provided that you do not try to update any other system in the same unit of work. If the first system you update in a unit of work is restricted, any attempt to update any other system in that unit of work returns an error.

Related concepts

Phase 6: Accessing data at a remote site (Db2 Installation and Migration)

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Remote servers and distributed data

Distributed data is data that resides on a database management system (DBMS) other than your local system. Your local DBMS is the one on which you bind your application plan. All other DBMSs are remote.

If you are requesting services from a remote DBMS, that DBMS is a server, and your local system is a requester or client.

Your application can be connected to many DBMSs at one time; the one that is currently performing work is the *current server*. When the local system is performing work, it also is called the current server.

A remote server can be physically remote, or it can be another subsystem of the same operating system that your local DBMS runs under. A remote server might be an instance of Db2 for z/OS, or it might be an instance of one of another product.

A DBMS, whether local or remote, is known to your Db2 system by its location name. The location name of a remote DBMS is recorded in the communications database.

Related tasks

Choosing names for the local subsystem (Db2 Installation and Migration)

Preparing for coordinated updates to two or more data sources

Two or more updates are coordinated if they must all commit or all roll back in the same unit of work.

About this task

This situation is common in banking. Suppose that an amount is subtracted from one account and added to another. The two actions must either both commit or both roll back at the end of the unit of work.

Procedure

Ensure that all systems that your program accesses implement two-phase commit processing. This processing ensures that updates to two or more DBMSs are coordinated automatically.

For example, Db2 and IMS, and Db2 and CICS, jointly implement a two-phase commit process. You can update an IMS database and a Db2 table in the same unit of work. If a system or communication failure occurs between committing the work on IMS and on Db2, the two programs restore the two systems to a consistent point when activity resumes.

You cannot do true coordinated updates within a DBMS that does not implement two-phase commit processing, because Db2 prevents you from updating such a DBMS and any other system within the same unit of work. In this context, update includes the statements INSERT, UPDATE, MERGE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, RENAME, COMMENT, and LABEL.

However, if you cannot implement two-phase commit processing on all systems that your program accesses, you can simulate the effect of coordinated updates by performing the following actions:

a. Update one system and commit that work.
b. Update the second system and commit its work.
c. Ensure that your program has code to undo the first update if a failure occurs after the first update is committed and before the second update is committed. No automatic provision exists for bringing the two systems back to a consistent point.

Related concepts

Two-phase commit process (Db2 Administration Guide)

Forcing restricted system rules in your program

A *restricted system* is a DBMS that does not implement two-phase commit processing. These systems have a number of update restrictions. You can restrict your program completely to the rules for these restricted systems, regardless of whether the program is accessing restricted systems or non-restricted systems.

About this task

Accessing restricted systems has the following limitations:

- For programs that access CICS or IMS, you cannot update data on restricted systems.
- Within a unit of work, you cannot update a restricted system after updating a non-restricted system.
- Within a unit of work, if you update a restricted system, you cannot update any other systems.

Procedure

When you prepare your program, specify the SQL processing option CONNECT(1).

This option applies type 1 CONNECT statement rules.

Restriction: Do not use packages that are precompiled with the CONNECT(1) option and packages that are precompiled with the CONNECT(2) option in the same package list. The first CONNECT statement that is executed by your program determines which rules are in effect for the entire execution: type 1 or type 2. If your program attempts to execute a later CONNECT statement that is precompiled with the other type, Db2 returns an error.

Related concepts

Options for SQL statement processing
Use SQL processing options to specify how the Db2 precompiler and the Db2 coprocessor interpret and process input, and how they present output.

Chapter 2. Connecting to Db2 from your application program

Application programs communicate with Db2 through an attachment facility. You must invoke an attachment facility, either implicitly or explicitly, before your program can interact with Db2.

About this task

You can use the following attachment facilities in a z/OS environment:

CICS attachment facility
Use this facility to access Db2 from CICS application programs.

IMS attachment facility
Use this facility to access Db2 from IMS application programs.

Time Sharing Option (TSO) attachment facility
Use this facility in a TSO or batch environment to communicate to a local Db2 subsystem. This facility invokes the DSN command processor.

Call attachment facility (CAF)
Use this facility as an alternative to the TSO attachment facility when your application needs tight control over the session environment.

Resource Recovery Services attachment facility (RRSAF)
Use this facility for stored procedures that run in a WLM-established address space or as an alternative to the CAF. RRSAF provides support for z/OS RRS as the recovery coordinator and supports other capabilities not present in CAF

For distributed applications, use the distributed data facility (DDF).

Requirement: Ensure that any application that requests Db2 services satisfies the following environment characteristics, regardless of the attachment facility that you use:

- The application must be running in TCB mode. SRB mode is not supported.
- An application task cannot have any Enabled Unlocked Task (EUT) functional recovery routines (FRRs) active when requesting Db2 services. If an EUT FRR is active, the Db2 functional recovery can fail, and your application can receive some unpredictable abends.
- Different attachment facilities cannot be active concurrently within the same address space. Specifically, the following requirements exist:
  - An application must not use CAF or RRSAF in an CICS or IMS address space.
  - An application that runs in an address space that has a CAF connection to Db2 cannot connect to Db2 by using RRSAF.
  - An application that runs in an address space that has an RRSAF connection to Db2 cannot connect to Db2 by using CAF.
  - An application cannot invoke the z/OS AXSET macro after executing the CAF CONNECT call and before executing the CAF DISCONNECT call.
- One attachment facility cannot start another. For example, your CAF or RRSAF application cannot use DSN, and a DSN RUN subcommand cannot call your CAF or RRSAF application.
- The language interface modules for CAF and RRSAF, DSNALI and DSNRLI, are shipped with the linkage attributes AMODE(31) and RMODE(ANY). If your applications load CAF or RRSAF below the 16-MB line, you must link-edit DSNALI or DSNRLI again.

Related concepts

Db2 attachment facilities (Introduction to Db2 for z/OS)
Distributed data facility (Introduction to Db2 for z/OS)

Invoking the call attachment facility

Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

Before you begin

Before you can invoke CAF, perform the following actions:

- Ensure that the CAF language interface (DSNALI) is available.
- Ensure that your application satisfies the requirements for programs that access CAF.
- Ensure that your application satisfies the general environment characteristics for connecting to Db2.
- Ensure that you are familiar with the following z/OS concepts and facilities:
  - The CALL macro and standard module linkage conventions
  - Program addressing and residency options (AMODE and RMODE)
  - Creating and controlling tasks; multitasking
  - Functional recovery facilities such as ESTAE, ESTAI, and FRRs
  - Asynchronous events and TSO attention exits (STAX)
  - Synchronization techniques such as WAIT/POST.

About this task

Applications that use CAF can be written in assembler language, C, COBOL, Fortran, and PL/I. When choosing a language to code your application in, consider the following restrictions:

- If you need to use z/OS macros (ATTACH, WAIT, POST, and so on), use a programming language that supports them or embed them in modules that are written in assembler language.
- The CAF TRANSLATE function is not available in Fortran. To use this function, code it in a routine that is written in another language, and then call that routine from Fortran.

Recommendations: For IMS and DSN applications, consider the following recommendations:

- For IMS batch applications, do not use CAF. Instead use the Db2 DL/I batch support. Although it is possible for IMS batch applications to access Db2 databases through CAF, that method does not coordinate the commitment of work between the IMS and Db2 systems.
- For DSN applications, do not use CAF unless you provide an application controller to manage the DSN application and replace any needed DSN functions. You might also have to change the application to communicate connection failures to the controller correctly. Running DSN applications with CAF is not advantageous, and the loss of DSN services can affect how well your program runs.

Procedure

Perform one of the following actions:

- Explicitly invoke CAF by including in your program CALL DSNALI statements with the appropriate options.

  The first option is a CAF connection function, which describes the action that you want CAF to take. The effect of any function depends in part on what functions the program has already run.

  Requirement: For C and PL/I applications, you must also include in your program the compiler directives that are listed in the following table, because DSNALI is an assembler language program.

Table 4. Compiler directives to include in C and PL/I applications that contain CALL DSNALI statements

Language	Compiler directive to include
C	`#pragma linkage(dsnali, OS)`
C++	`extern "OS" { int DSNALI( char * functn, ...); }`
PL/I	`DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE;`

- Implicitly invoke CAF by including SQL statements or IFI calls in your program just as you would in any program. The CAF facility establishes the connections to Db2 with the default values for the subsystem name and plan name.

  Restriction: If your program can make its first SQL call from different modules with different DBRMs, you cannot use a default plan name and thus, you cannot implicitly invoke CAF. Instead, you must explicitly invoke CAF by using the OPEN function.

  Requirement: If your application includes both SQL and IFI calls, you must issue at least one SQL call before you issue any IFI calls. This action ensures that your application uses the correct plan.

  Although doing so is not recommended, you can run existing DSN applications with CAF by allowing them to make implicit connections to Db2. For Db2 to make an implicit connection successfully, the plan name for the application must be the same as the member name of the database request module (DBRM) that Db2 produced when you precompiled the source program that contains the first SQL call. You must also substitute the DSNALI language interface module for the TSO language interface module, DSNELI.

If you do not specify the return code and reason code parameters in your CAF calls or you invoked CAF implicitly, CAF puts a return code in register 15 and a reason code in register 0.

To determine if an implicit connection was successful, the application program should examine the return and reason codes immediately after the first executable SQL statement in the application program by performing one of the following actions:

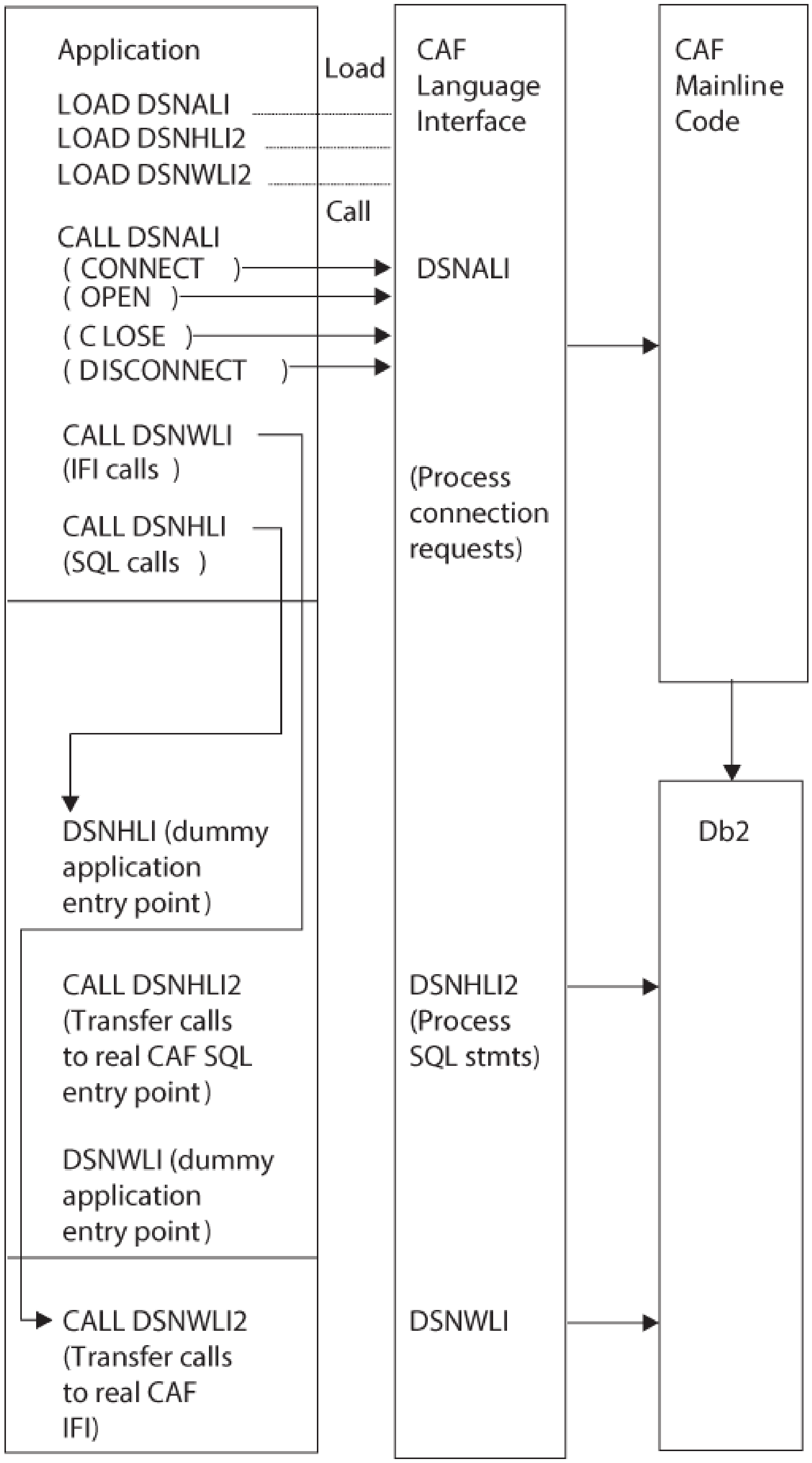
- Examining registers 0 and 15 directly.
- Examining the SQLCA, and if the SQLCODE is -991, obtain the return and reason code from the message text. The return code is the first token, and the reason code is the second token.

If the implicit connection was successful, the application can examine the SQLCODE for the first, and subsequent, SQL statements.

Examples

Example of a CAF configuration

The following figure shows an conceptual example of invoking and using CAF. The application contains statements to load DSNALI, DSNHLI2, and DSNWLI2. The application accesses Db2 by using the CAF Language Interface. It calls DSNALI to handle CAF requests, DSNWLI to handle IFI calls, and DSNHLI to handle SQL calls.



Sample programs that use CAF

You can find a sample assembler program (DSN8CA) and a sample COBOL program (DSN8CC) that use the CAF in library *prefix*.SDSNSAMP. A PL/I application (DSN8SPM) calls DSN8CA, and a COBOL application (DSN8SCM) calls DSN8CC.

Related concepts

Sample applications supplied with Db2 for z/OS

Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

Related reference

CAF connection functions
A CAF connection function specifies the action that you want CAF to take. You specify these functions when you invoke CAF through CALL DSNALI statements.

Call attachment facility

An attachment facility enables programs to communicate with Db2. The call attachment facility (CAF) provides such a connection for programs that run in z/OS batch, TSO foreground, and TSO background. The CAF needs tight control over the session environment.

A program that uses CAF can perform the following actions:

- Access Db2 from z/OS address spaces where TSO, IMS, or CICS do not exist.
- Access Db2 from multiple z/OS tasks in an address space.
- Access the Db2 IFI.
- Run when Db2 is down.

  Restriction: The application cannot run SQL when Db2 is down.
- Run with or without the TSO terminal monitor program (TMP).
- Run without being a subtask of the DSN command processor or of any Db2 code.
- Run above or below the 16-MB line. (The CAF code resides below the line.)
- Establish an explicit connection to Db2, through a CALL interface, with control over the exact state of the connection.
- Establish an implicit connection to Db2, by using SQL statements or IFI calls without first calling CAF, with a default plan name and subsystem identifier.
- Verify that the application is using the correct release of Db2.
- Supply event control blocks (ECBs), for Db2 to post, that signal startup or termination.
- Intercept return codes, reason codes, and abend codes from Db2 and translate them into messages.

Any task in an address space can establish a connection to Db2 through CAF. Only one connection can exist for each task control block (TCB). A Db2 service request that is issued by a program that is running under a given task is associated with that task's connection to Db2. The service request operates independently of any Db2 activity under any other task.

Each connected task can run a plan. Multiple tasks in a single address space can specify the same plan, but each instance of a plan runs independently from the others. A task can terminate its plan and run a different plan without fully breaking its connection to Db2.

CAF does not generate task structures.

When you design your application, consider that using multiple simultaneous connections can increase the possibility of deadlocks and Db2 resource contention.

A tracing facility provides diagnostic messages that aid in debugging programs and diagnosing errors in the CAF code. In particular, attempts to use CAF incorrectly cause error messages in the trace stream.

Restriction: CAF does not provide attention processing exits or functional recovery routines. You can provide whatever attention handling and functional recovery your application needs, but you must use ESTAE/ESTAI type recovery routines and not Enabled Unlocked Task (EUT) FRR routines.

Properties of CAF connections

Call attachment facility (CAF) enables programs to communicate with Db2.

The connection that CAF makes with Db2 has the basic properties that are listed in the following table.

Table 5. Properties of CAF connections

Property	Value	Comments
Connection name	DB2CALL	You can use the DISPLAY THREAD command to list CAF applications that have the connection name DB2CALL.
Connection type	BATCH	BATCH connections use a single phase commit process that is coordinated by Db2. Application programs can also control when statements are committed by using the SQL COMMIT and ROLLBACK statements.
Authorization IDs	Authorization IDs that are associated with the address space	Db2 establishes authorization IDs for each task's connection when it processes that connection. For the BATCH connection type, Db2 creates a list of authorization IDs based on the authorization ID that is associated with the address space. This list is the same for every task. A location can provide a Db2 connection authorization exit routine to change the list of IDs.
Scope	CAF processes connections as if each task is entirely isolated. When a task requests a function, the CAF passes the functions to Db2 and is unaware of the connection status of other tasks in the address space. However, the application program and the Db2 subsystem are aware of the connection status of multiple tasks in an address space.	none

If a connected task terminates normally before the CLOSE function deallocates the plan, Db2 commits any database changes that the thread made since the last commit point. If a connected task abends before the CLOSE function deallocates the plan, Db2 rolls back any database changes since the last commit point. In either case, Db2 deallocates the plan, if necessary, and terminates the task's connection before it allows the task to terminate.

If Db2 abnormally terminates while an application is running, the application is rolled back to the last commit point. If Db2 terminates while processing a commit request, Db2 either commits or rolls back any changes at the next restart. The action taken depends on the state of the commit request when Db2 terminates.

Related concepts

Connection routines and sign-on routines (Managing Security)

Attention exit routines for CAF

An attention exit routine enables you to regain control from Db2 during long-running or erroneous requests. Call attachment facility (CAF) has no attention exit routines, but you can provide your own if necessary.

An attention exit routine works by detaching the TCB that is currently waiting on an SQL or IFI request to complete. After the TCB is detached, Db2 detects the resulting abend and performs termination processing for that task. The termination processing includes any necessary rollback of transactions.

You can provide your own attention exit routines. However, your routine might not get control if you request attention while Db2 code is running, because Db2 uses enabled unlocked task (EUT) functional recovery routines (FRRs).

Recovery routines for CAF

You can use abend recovery routines and functional recovery routines (FRRs) to handle unexpected errors. An abend recovery routine controls what happens when an abend occurs while Db2 has control. A functional recovery routine can obtain information about and recover from program errors.

The CAF has no abend recovery routines, but you can provide your own. Any abend recovery routines that you provide must use tracking indicators to determine if an abend occurred during Db2 processing. If an abend occurs while Db2 has control, the recovery routine can take one of the following actions:

- Allow task termination to complete. Do not try the program again. Db2 detects task termination and terminates the thread with the ABRT parameter. You lose all database changes back to the last sync point or commit point.

  This action is the only action that you can take for abends that are caused by the CANCEL command or by DETACH. You cannot use additional SQL statements. If you attempt to execute another SQL statement from the application program or its recovery routine, you receive a return code of +256 and a reason code of X'00F30083'.
- In an ESTAE routine, issue a CLOSE function call with the ABRT parameter followed by a DISCONNECT function call. The ESTAE exit routine can try again so that you do not need to reinstate the application task.

FRRs must comply with the following requirements and restrictions:

- You can use only enabled unlocked task (EUT) FRRs in your routines that call Db2. The standard z/OS functional recovery routines (FRRs) apply to only code that runs in service request block (SRB) mode, and Db2 does not support calls from SRB mode routines.
- Do not have an EUT FRR active when using CAF, processing SQL requests, or calling IFI. With z/OS, if you have an active EUT FRR, all Db2 requests fail, including the initial CONNECT or OPEN request. The requests fail because Db2 always creates an ARR-type ESTAE, and z/OS does not allow the creation of ARR-type ESTAEs when an FRR is active.
- An EUT FRR cannot retry failing Db2 requests. An EUT FRR retry bypasses ESTAE routines from Db2. The next Db2 request of any type, including a DISCONNECT request, fails with a return code of +256 and a reason code of X'00F30050'.

Making the CAF language interface (DSNALI) available

Before you can invoke the call attachment facility (CAF), you must first make DSNALI available.

About this task

Part of CAF is a Db2 load module, DSNALI, which is also known as the CAF language interface. DSNALI has the alias names DSNHLI2 and DSNWLI2. The module has five entry points: DSNALI, DSNHLI, DSNHLI2, DSNWLI, and DSNWLI2. These entry points serve the following functions:

- Entry point DSNALI handles explicit Db2 connection service requests.

- DSNHLI and DSNHLI2 handle SQL calls. Use DSNHLI if your application program link-edits DSNALI. Use DSNHLI2 if your application program loads DSNALI.
- DSNWLI and DSNWLI2 handle IFI calls. Use DSNWLI if your application program link-edits DSNALI. Use DSNWLI2 if your application program loads DSNALI.

Procedure

To make DSNALI available:

1. Decide which of the following methods you want to use to make DSNALI available:
   - Explicitly issuing LOAD requests when your program runs.

     By explicitly loading the DSNALI module, you beneficially isolate the maintenance of your application from future IBM maintenance to the language interface. If the language interface changes, the change will probably not affect your load module.
   - Including the DSNALI module in your load module when you link-edit your program.

     If you do not need explicit calls to DSNALI for CAF functions, link-editing DSNALI into your load module has some advantages. When you include DSNALI during the link-edit, you do not need to code a dummy DSNHLI entry point in your program or specify the precompiler option ATTACH. Module DSNALI contains an entry point for DSNHLI, which is identical to DSNHLI2, and an entry point DSNWLI, which is identical to DSNWLI2.

     A disadvantage to link-editing DSNALI into your load module is that any IBM maintenance to DSNALI requires a new link-edit of your load module.

   Alternatively, if using explicit connections via CALL DSNALI, you can link-edit your program with DSNULI, the Universal Language Interface.
2. Depending on the method that you chose in step 1, perform one of the following actions:
   - **If you want to explicitly issue LOAD requests when your program runs:**

     In your program, issue z/OS LOAD service requests for entry points DSNALI and DSNHLI2. If you use IFI services, you must also load DSNWLI2. The entry point addresses that LOAD returns are saved for later use with the CALL macro. Indicate to Db2 which entry point to use in one of the following two ways:
     - Specify the precompiler option ATTACH(CAF).

       This option causes Db2 to generate calls that specify entry point DSNHLI2.

       Restriction: You cannot use this option if your application is written in Fortran.
     - Code a dummy entry point named DSNHLI within your load module.

       If you do not specify the precompiler option ATTACH, the Db2 precompiler generates calls to entry point DSNHLI for each SQL request. The precompiler does not know about and is independent of the different Db2 attachment facilities. When the calls generated by the Db2 precompiler pass control to DSNHLI, your code that corresponds to the dummy entry point must preserve the option list that was passed in R1 and specify the same option list when it calls DSNHLI2.
   - **If you want to include the DSNALI module in your load module when you link-edit your program:**

     Include DSNALI in your load module during a link-edit step. The module must be in a load module library, which is included either in the SYSLIB concatenation or another INCLUDE library that is defined in the linkage editor JCL. Because all language interface modules contain an entry point declaration for DSNHLI, the linkage editor JCL must contain an INCLUDE linkage editor control statement for DSNALI; for example, `INCLUDE SYSLIB(DSNALI)`. By coding these options, you avoid inadvertently picking up the wrong language interface module.

Related concepts

LOB file reference variables

In a host application, you can use a file reference variable to insert a LOB or XML value from a file into a Db2 table. You can also use a file reference variable to select a LOB or XML value from a Db2 table into a file.

Examples of invoking CAF
The call attachment facility (CAF) enables programs to communicate with Db2. If you explicitly invoke CAF in your program, you can use the CAF connection functions to control the state of the connection.

"Universal language interface (DSNULI)" on page 117
The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

Related tasks

Link-editing an application with DSNULI
To create a single load module that can be used in more than one attachment environment, you can link-edit your program or stored procedure with the Universal Language Interface module (DSNULI) instead of with one of the environment-specific language interface modules (DSNELI, DSNALI, DSNRLI, or DSNCLI).

Saving storage when manipulating LOBs by using LOB locators
LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. By using locators, you avoid needing to allocate the large amounts of storage that are needed for host variables to hold LOB data.

Requirements for programs that use CAF

The call attachment facility (CAF) enables programs to communicate with Db2. Before you invoke CAF in your program, ensure that your program satisfies any requirements for using CAF.

When you write programs that use CAF, ensure that they meet the following requirements:

- The program accounts for the size of the CAF code. The CAF code requires about 16 KB of virtual storage per address space and an additional 10 KB for each TCB that uses CAF.
- If your local environment intercepts and replaces the z/OS LOAD SVC that CAF uses, you must ensure that your version of LOAD manages the load list element (LLE) and contents directory entry (CDE) chains like the standard z/OS LOAD macro. CAF uses z/OS SVC LOAD to load two modules as part of the initialization after your first service request. Both modules are loaded into fetch-protected storage that has the job-step protection key.
- If you use CAF from IMS batch, you must write data to only one system in any one unit of work. If you write to both systems within the same unit, a system failure can leave the two databases inconsistent with no possibility of automatic recovery. To end a unit of work in Db2, execute the SQL COMMIT statement. To end a unit of work in IMS, issue the SYNCPOINT command.

You can prepare application programs to run in CAF similar to how you prepare applications to run in other environments, such as CICS, IMS, and TSO. You can prepare a CAF application either in the batch environment or by using the Db2 program preparation process. You can use the program preparation system either through DB2I or through the DSNH CLIST.

Related tasks

Preparing an application to run on Db2 for z/OS
To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

How CAF modifies the content of registers

If you do not specify the return code and reason code parameters in your CAF function calls or if you invoke CAF implicitly, CAF puts a return code in register 15 and a reason code in register 0. The contents of registers 2 through 14 are preserved across calls.

The following table lists the standard calling conventions for registers R1, R13, R14, and R15.

Table 6. Standard usage of registers R1, R13, R14, and R15

Register	Usage
R1	CALL DSNALI parameter list pointer
R13	Address of caller's save area
R14	Caller's return address
R15	CAF entry point address

Your CAF program should respect these register conventions.

CAF also supports high-level languages that cannot examine the contents of individual registers.

Related concepts

CALL DSNALI statement parameter list
The CALL DSNALI statement explicitly invokes CAF. When you include CALL DSNALI statements in your program, you must specify all parameters that come before the return code parameter.

Implicit connections to CAF

If the CAF language interface (DSNALI) is available and you do not explicitly specify CALL DSNALI statements in your application, CAF initiates implicit CONNECT and OPEN requests to Db2. These requests are subject to the same Db2 return codes and reason codes as explicitly specified requests.

Implicit connections use the following defaults:

Subsystem name

The default name that is specified in the module DSNHDECP. CAF uses the installation default DSNHDECP, unless your own DSNHDECP module is in a library in a STEPLIB statement of a JOBLIB concatenation or in the link list. In a data sharing group, the default subsystem name is the group attachment name.

Implicit connections to CAF always use DSNHDECP as the user-specified application defaults module.

Be certain that you know what the default name is and that it names the specific Db2 subsystem you want to use.

Plan name

The member name of the database request module (DBRM) that Db2 produced when you precompiled the source program that contains the first SQL call.

Different types of implicit connections exist. The simplest is for an application to call neither the CONNECT nor OPEN functions. You can also use the CONNECT function only or the OPEN function only. Each of these calls implicitly connects your application to Db2. To terminate an implicit connection, you must use the proper calls.

Related concepts

Summary of CAF behavior
The effect of any CAF function depends in part on what functions the program has already run. You should plan the CAF function calls that your program makes to avoid any errors and major structural problems in your application.

CALL DSNALI statement parameter list

The CALL DSNALI statement explicitly invokes CAF. When you include CALL DSNALI statements in your program, you must specify all parameters that come before the return code parameter.

For CALL DSNALI statements, use a standard z/OS CALL parameter list. Register 1 points to a list of fullword addresses that point to the actual parameters. The last address must contain a 1 in the high-order bit.

In CALL DSNALI statements, you cannot omit any of parameters that come before the return code parameter by coding zeros or blanks. No defaults exist for those parameters for explicit connection requests. Defaults are provided for only implicit connections. All parameters starting with the return code parameter are optional.

When you want to use the default value for a parameter but specify subsequent parameters, code the CALL DSNALI statement as follows:

- For C-language, when you code CALL DSNALI statements in C, you need to specify the address of every required parameter, using the "address of" operator (&), and not the parameter itself. For example, to pass the *startecb* parameter on CONNECT, specify the address of the 4-byte integer (&secb).

```
char functn[13] = "CONNECT     ";
char   ssid[5] = "DB2A";
int    tecb     = 0;
int    secb     = 0;
ptr    ribptr;
int    retcode;
int    reascode;
ptr    eibptr;

fnret = dsnali(&functn[0], &ssid[0], &tecb, &secb, &ribptr, &retcode, &reascode,
               NULL, &eibptr);
```

- For other languages except assembler language, code zero for that parameter in the CALL DSNALI statement. For example, suppose that you are coding a CONNECT call in a COBOL program, and you want to specify all parameters except the return code parameter. You can write a statement similar to the following statement:

```
CALL 'DSNALI' USING FUNCTN SSID TECB SECB RIBPTR
   BY CONTENT ZERO BY REFERENCE REASCODE SRDURA EIBPTR.
```

- For assembler language, code a comma for that parameter in the CALL DSNALI statement. For example, to specify all optional parameters except the return code parameter write a statement similar to the following statement:

```
CALL DSNALI,(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,,REASCODE,SRDURA,EIBPTR,
GROUPOVERRIDE)
```

The following figure shows a sample parameter list structure for the CONNECT function.

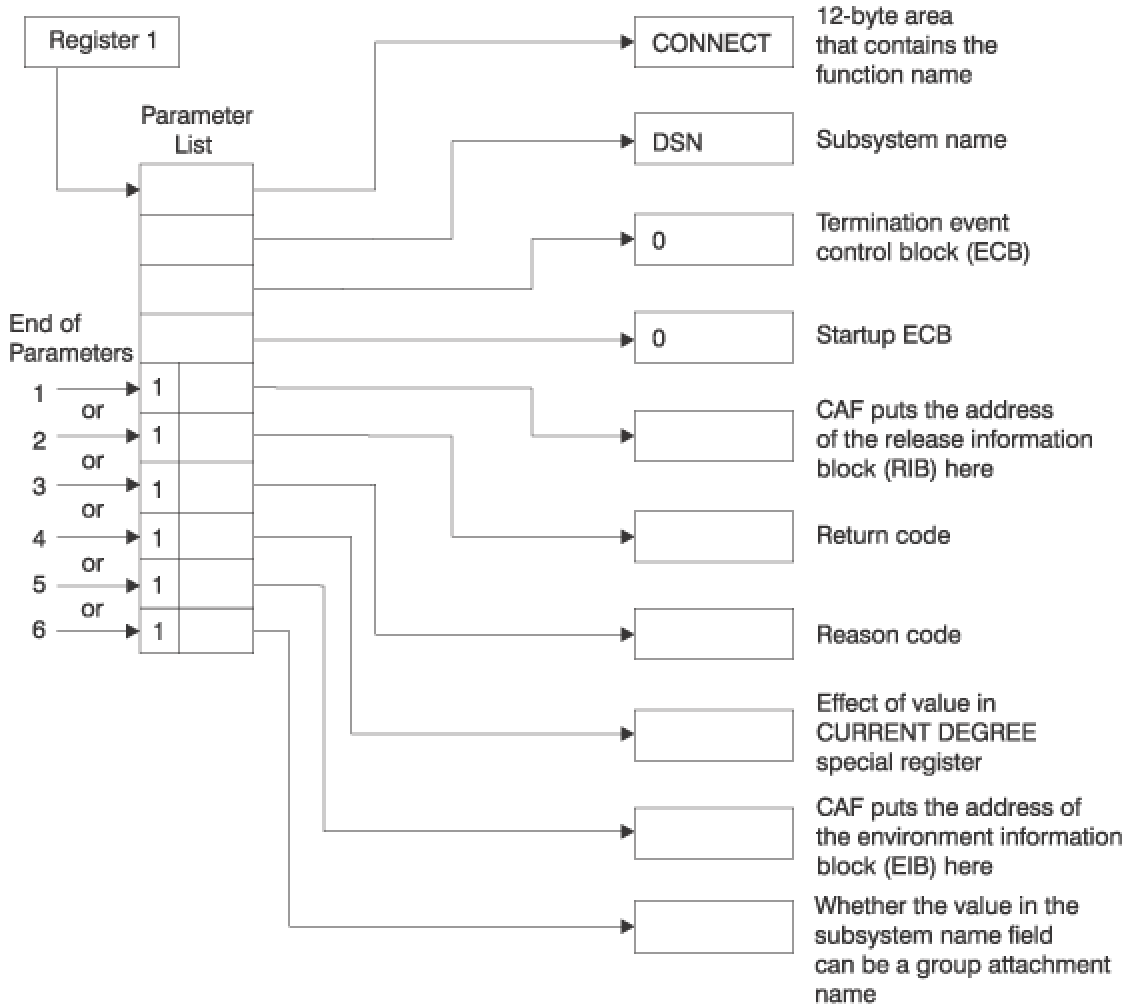


Figure 1. The parameter list for a CONNECT call

The preceding figure illustrates how you can omit parameters for the CALL DSNALI statement to control the return code and reason code fields after a CONNECT call. You can terminate the parameter list at any of the following points. These termination points apply to all CALL DSNALI statement parameter lists.

1. Terminates the parameter list without specifying the parameters *retcode*, *reascode*and *srdura* and places the return code in register 15 and the reason code in register 0.

   Terminating the parameter list at this point ensures compatibility with CAF programs that require a return code in register 15 and a reason code in register 0.
2. Terminates the parameter list after the parameter *retcode* and places the return code in the parameter list and the reason code in register 0.

   Terminating the parameter list at this point enables the application program to take action, based on the return code, without further examination of the associated reason code.
3. Terminates the parameter list after the parameter *reascode* and places the return code and the reason code in the parameter list.

   Terminating the parameter list at this point provides support to high-level languages that are unable to examine the contents of individual registers.

   If you code your CAF application in assembler language, you can specify the reason code parameter and omit the return code parameter.
4. Terminates the parameter list after the parameter *srdura*.

   If you code your CAF application in assembler language, you can specify this parameter and omit the *retcode* and *reascode* parameters.

5. Terminates the parameter list after the parameter *eibptr*.

   If you code your CAF application in assembler language, you can specify this parameter and omit the *retcode*, *reascode*, or *srdura* parameters.

6. Terminates the parameter list after the parameter *groupoverride*.

   If you code your CAF application in assembler language, you can specify this parameter and omit the *retcode*, *reascode*,*srdura*, or *eibptr* parameters.

Even if you specify that the return code be placed in the parameter list, it is also placed in register 15 to accommodate high-level languages that support special return code processing.

Related concepts

How CAF modifies the content of registers
If you do not specify the return code and reason code parameters in your CAF function calls or if you invoke CAF implicitly, CAF puts a return code in register 15 and a reason code in register 0. The contents of registers 2 through 14 are preserved across calls.

Summary of CAF behavior

The effect of any CAF function depends in part on what functions the program has already run. You should plan the CAF function calls that your program makes to avoid any errors and major structural problems in your application.

The following table summarizes CAF behavior after various inputs from application programs. The top row lists the possible CAF functions that programs can call. The first column lists the task's most recent history of connection requests. For example, the value "CONNECT followed by OPEN" in the first column means that the task issued CONNECT and then OPEN with no other CAF calls in between. The intersection of a row and column shows the effect of the next call if it follows the corresponding connection history. For example, if the call is OPEN and the connection history is CONNECT, the effect is OPEN; the OPEN function is performed. If the call is SQL and the connection history is empty (meaning that the SQL call is the first CAF function the program), the effect is that implicit CONNECT and OPEN functions are performed, followed by the SQL function.

Table 7. Effects of CAF calls, as dependent on connection history

Previous function	Next function					
	CONNECT	**OPEN**	**SQL**	**CLOSE**	**DISCONNECT**	**TRANSLATE**
Empty: first call	CONNECT	OPEN	CONNECT, OPEN, followed by the SQL or IFI call	Error 203[1]	Error 204[1]	Error 205[1]
CONNECT	Error 201[1]	OPEN	OPEN, followed by the SQL or IFI call	Error 203[1]	DISCONNECT	TRANSLATE
CONNECT followed by OPEN	Error 201[1]	Error 202[1]	The SQL or IFI call	CLOSE[2]	DISCONNECT	TRANSLATE
CONNECT followed by SQL or IFI call	Error 201[1]	Error 202[1]	The SQL or IFI call	CLOSE[2]	DISCONNECT	TRANSLATE
OPEN	Error 201[1]	Error 202[1]	The SQL or IFI call	CLOSE[2]	Error 204[1]	TRANSLATE
SQL or IFI call	Error 201[1]	Error 202[1]	The SQL or IFI call	CLOSE[2]	Error 204[1]	TRANSLATE[3]

Notes:

1. An error is shown in this table as Error *nnn*. The corresponding reason code is X'00C10*nnn*'. The message number is DSNA*nnn*I or DSNA*nnn*E.
2. The task and address space connections remain active. If the CLOSE call fails because Db2 was down, the CAF control blocks are reset, the function produces return code 4 and reason code X'00C10824', and CAF is ready for more connection requests when Db2 is up.
3. A TRANSLATE request is accepted, but in this case it is redundant. CAF automatically issues a TRANSLATE request when an SQL or IFI request fails.

Related reference

CAF return codes and reason codes
CAF provides the return codes either to the corresponding parameters that are specified in a CAF function call or, if you choose not to use those parameters, to registers 15 and 0.

CAF connection functions

A CAF connection function specifies the action that you want CAF to take. You specify these functions when you invoke CAF through CALL DSNALI statements.

You can specify the following CAF functions in a CALL DSNALI statement:

CONNECT
Establishes the task (TCB) as a user of the named Db2 subsystem. When the first task within an address space issues a connection request, the address space is also initialized as a user of Db2.

OPEN
Allocates a Db2 plan. You must allocate a plan before Db2 can process SQL statements. If you did not request the CONNECT function, the OPEN function implicitly establishes the task, and optionally the address space, as a user of Db2.

CLOSE
Commits or abnormally terminates any database changes and deallocates the plan. If the OPEN function implicitly requests the CONNECT function, the CLOSE function removes the task, and possibly the address space, as a user of Db2.

DISCONNECT
Removes the task as a user of Db2 and, if this task is the last or only task in the address space with a Db2 connection, terminates the address space connection to Db2.

TRANSLATE
Returns an SQL code and printable text that describe a Db2 hexadecimal error reason code. This information is returned to the SQLCA.

Restriction: You cannot call the TRANSLATE function from the Fortran language.

Recommendation: Because the effect of any CAF function depends on what functions the program has already run, carefully plan the calls that your program makes to these CAF connection functions. Read about the summary of CAF behavior and make these function calls accordingly.

Related concepts

Summary of CAF behavior
The effect of any CAF function depends in part on what functions the program has already run. You should plan the CAF function calls that your program makes to avoid any errors and major structural problems in your application.

CALL DSNALI statement parameter list

The CALL DSNALI statement explicitly invokes CAF. When you include CALL DSNALI statements in your program, you must specify all parameters that come before the return code parameter.

CONNECT function for CAF

The CAF CONNECT function initializes a connection to Db2. This function is different than the SQL CONNECT statement that accesses a remote location within Db2.

The CONNECT function establishes the caller's task as a user of Db2 services. If no other task in the address space currently holds a connection with the specified subsystem, the CONNECT function also initializes the address space for communication to the Db2 address spaces. The CONNECT function establishes the address space's cross memory authorization to Db2 and builds address space control blocks. You can issue a CONNECT request from any or all tasks in the address space, but the address space level is initialized only once when the first task connects.

Using the CONNECT function is optional. If you do not call the CONNECT function, the first request from a task, either an OPEN request or an SQL or IFI call, causes CAF to issue an implicit CONNECT request. If a task is connected implicitly, the connection to Db2 is terminated either when you call the CLOSE function or when the task terminates.

Call the CONNECT function in all of the following situations:

- You need to specify a particular subsystem name (*ssnm*) other than the default subsystem name.
- You need the value of the CURRENT DEGREE special register to last as long as the connection (*srdura*).
- You need to monitor the Db2 startup ECB (*startecb*), the Db2 termination ECB (*termecb*), or the Db2 release level.
- You plan to have multiple tasks in the address space open and close plans or a single task in the address space open and close plans more than once.

  Establishing task and address space level connections involves significant overhead. Using the CONNECT function to establish a task connection explicitly minimizes this overhead by ensuring that the connection to Db2 remains after the CLOSE function deallocates a plan. In this case, the connection terminates only when you use the DISCONNECT function or when the task terminates.

The CONNECT function also enables the caller to learn the following items:

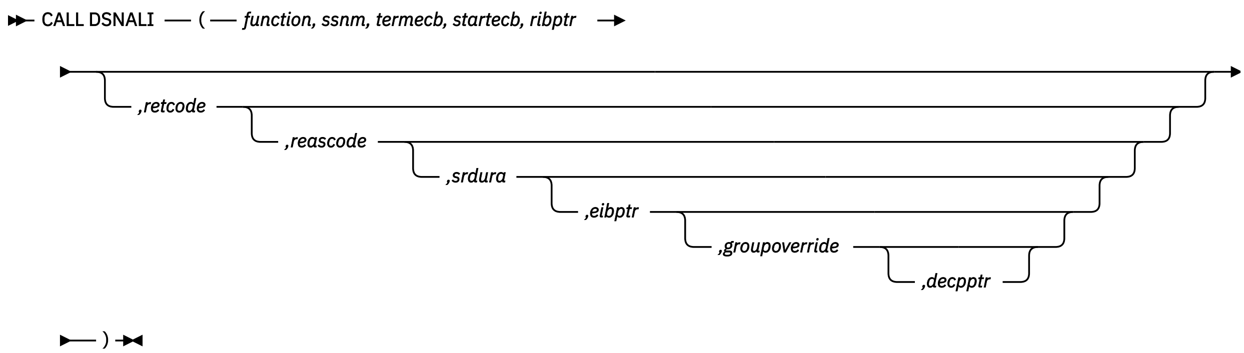
- That the operator has issued a STOP DB2 command. When this event occurs, Db2 posts the termination ECB, *termecb*. Your application can either wait on or just look at the ECB.
- That Db2 is abnormally terminating. When this event occurs happens, Db2 posts the termination ECB, *termecb*.
- That Db2 is available again after a connection attempt that failed because Db2 was down. Your application can either wait or look at the startup ECB, *startecb*. Db2 ignores this ECB if it was active at the time of the CONNECT request.
- The current release level of Db2. To find this information, access the RIBREL field in the release information block (RIB). If RIBREL is '999', the actual version, release, and modification level of Db2 is indicated in the RIBRELX field and its subfields.

Restriction: Do not issue CONNECT requests from a TCB that already has an active Db2 connection.

Recommendation: Do not mix explicit CONNECT and OPEN requests with implicitly established connections in the same address space. Either explicitly specify which Db2 subsystem you want to use or allow all requests to use the default subsystem.

The following diagram shows the syntax for the CONNECT function.

DSNALI CONNECT function



Parameters point to the following areas:

function
A 12-byte area that contains CONNECT followed by five blanks.

ssnm
A 4-byte Db2 subsystem name or group attachment or subgroup attachment name (if used in a data sharing group) to which the connection is made.

If *ssnm* is less than four characters long, pad it on the right with blanks to a length of four characters.

termecb
A 4-byte integer representing the application's event control block (ECB) for Db2 termination. Db2 posts this ECB when the operator enters the STOP DB2 command or when Db2 is abnormally terminating. The ECB indicates the type of termination by a POST code, as shown in the following table:

Table 8. POST codes and related termination types

POST code	Termination type
8	QUIESCE
12	FORCE
16	ABTERM

Before you check *termecb* in your CAF application program, first check the return code and reason code from the CONNECT call to ensure that the call completed successfully.

startecb
A 4-byte integer representing the application's startup ECB. If Db2 has not yet started when the application issues the call, Db2 posts the ECB when it successfully completes its startup processing. Db2 posts at most one startup ECB per address space. The ECB is the one associated with the most recent CONNECT call from that address space. Your application program must examine any nonzero CAF and Db2 reason codes before issuing a WAIT on this ECB.

If *ssnm* is a group attachment or subgroup attachment name, the first Db2 subsystem that starts on the local z/OS system and matches the specified group attachment name posts the ECB.

ribptr
A 4-byte area in which CAF places the address of the release information block (RIB) after the call. You can determine what release level of Db2 you are currently running by examining the RIBREL field. If RIBREL is '999', the actual version, release, and modification level of Db2 is indicated in the RIBRELX field and its subfields.You can determine the modification level within the release level by examining the RIBCNUMB and RIBCINFO fields. If the value in the RIBCNUMB field is greater than zero, check the RIBCINFO field for modification levels.

If the RIB is not available (for example, if you name a subsystem that does not exist), Db2 sets the 4-byte area to zeros.

The area to which *ribptr* points is below the 16-MB line.

Your program does not have to use the release information block, but it cannot omit the *ribptr* parameter.

Macro DSNDRIB maps the release information block (RIB). It can be found in *prefix*.SDSNMACS(DSNDRIB).

retcode
A 4-byte area in which CAF places the return code.

This field is optional. If you do not specify *retcode*, CAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which CAF places a reason code.

This field is optional. If you do not specify *reascode*, CAF places the reason code in register 0. If you specify *reascode*, you must also specify *retcode*.

srdura
A 10-byte area that contains the string 'SRDURA(CD)'. This field is optional. If you specify *srdura*, the value in the CURRENT DEGREE special register stays in effect from the time of the CONNECT call until the time of the DISCONNECT call. If you do not specify *srdura*, the value in the CURRENT DEGREE special register stays in effect from the time of the OPEN call until the time of the CLOSE call. If you specify this parameter in any language except assembler, you must also specify *retcode* and *reascode*. In assembler language, you can omit these parameters by specifying commas as placeholders.

eibptr
A 4-byte area in which CAF puts the address of the environment information block (EIB). The EIB contains information that you can use if you are connecting to a Db2 subsystem that is part of a data sharing group. For example, you can determine the name of the data sharing group, the member to which you are connecting, and whether the subsystem is in new-function mode. If the Db2 subsystem that you connect to is not part of a data sharing group, the fields in the EIB that are related to data sharing are blank. If the EIB is not available (for example, if you name a subsystem that does not exist), Db2 sets the 4-byte area to zeros.

The area to which *eibptr* points is above the 16-MB line.

You can omit this parameter when you make a CONNECT call.

If you specify this parameter in any language except assembler, you must also specify *retcode*, *reascode*, and *srdura*. In assembler language, you can omit *retcode*, *reascode*, and *srdura* by specifying commas as placeholders.

Macro DSNDEIB maps the EIB. It can be found in *prefix*.SDSNMACS(DSNDEIB).

groupoverride
An 8-byte area that the application provides. This parameter is optional. If you do not want group attach to be attempted, specify 'NOGROUP'. This string indicates that the subsystem name that is specified by *ssnm* is to be used as a Db2 subsystem name, even if *ssnm* matches a group attachment or subgroup attachment name. If *groupoverride* is not provided, *ssnm* is used as the group attachment or subgroup attachment name if it matches a group attachment or subgroup attachment name.

If you specify this parameter in any language except assembler, you must also specify *retcode*, *reascode*, *srdura*, and *eibptr*. In assembler language, you can omit *retcode*, *reascode*, *srdura*, and *eibptr* by specifying commas as placeholders.

Recommendation: Avoid using the *groupoverride* parameter when possible, because it limits the ability to do dynamic workload routing in a Parallel Sysplex®. However, you should use this parameter in a data sharing environment when you want to connect to a specific member of a data sharing group, and the subsystem name of that member is the same as the group attachment or subgroup attachment name.

decpptr

A 4-byte area in which CAF is to put the address of the DSNHDECP control block or user-specified application defaults module that was loaded by subsystem *ssnm* when that subsystem was started. This 4-byte area is a 31-bit pointer. If *ssnm* is not found, the 4-byte area is set to 0.

The area to which decpptr points may be above the 16-MB line.

If you specify this parameter in any language except assembler, you must also specify the *retcode*, *reascode*, *srdura*, *eibptr*, and *groupoverride* parameters. In assembler language, you can omit the *retcode*, *reascode*, *srdura*, *eibptr*, and *groupoverride* parameters by specifying commas as placeholders.

Example of CAF CONNECT function calls

The following table shows a CONNECT call in each language.

Table 9. Examples of CAF CONNECT function calls

Language	Call example
Assembler	`CALL DSNALI,(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR, GRPOVER)`
C[1]	`fnret=dsnali(&functn[0],&ssid[0], &tecb, &secb,&ribptr,&retcode, &reascode, &srdura[0], &eibptr, &grpover[0]);`
COBOL	`CALL 'DSNALI' USING FUNCTN SSID TERMECB STARTECB RIBPTR RETCODE REASCODE SRDURA EIBPTR GRPOVER.`
Fortran	`CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR,GRPOVER)`
PL/I[1]	`CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR,GRPOVER)`

Note:

- For C and PL/I applications, you must include the appropriate compiler directives, because DSNALI is an assembler language program. These compiler directives are described in the instructions for invoking CAF.

Related concepts

Examples of invoking CAF
The call attachment facility (CAF) enables programs to communicate with Db2. If you explicitly invoke CAF in your program, you can use the CAF connection functions to control the state of the connection.

Related tasks

Invoking the call attachment facility
Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

Related reference

Synchronizing Tasks (WAIT, POST, and EVENTS Macros) (MVS Programming: Assembler Services Guide)

OPEN function for CAF

The OPEN function allocates Db2 resources that are needed to run the specified plan or to issue IFI requests. If the requesting task does not already have a connection to the named Db2 subsystem, the OPEN function establishes it.

Using the OPEN function is optional. If you do not call the OPEN function, the actions that the OPEN function perform occur implicitly on the first SQL or IFI call from the task.

Restriction: Do not use the OPEN function if the task already has a plan allocated.

The following diagram shows the syntax for the OPEN function.

DSNALI OPEN function

```
>>- CALL DSNALI -- ( -- function, ssnm, plan  ->

 >--+------------------------------------------------+-- ) -><
    '-- , -- retcode --+----------------------------+'
                       '-- , -- reascode --+-------------------------+'
                                           '-- , -- groupoverride --'
```

Parameters point to the following areas:

function
: A 12-byte area that contains the word OPEN followed by eight blanks.

ssnm
: A 4-byte Db2 subsystem name or group attachment or subgroup attachment name (if used in a data sharing group). The OPEN function allocates the specified plan to this Db2 subsystem. Also, if the requesting task does not already have a connection to the named Db2 subsystem, the OPEN function establishes it.

  You must specify the *ssnm* parameter, even if the requesting task also issues a CONNECT call. If a task issues a CONNECT call followed by an OPEN call, the subsystem names for both calls must be the same.

  If *ssnm* is less than four characters long, pad it on the right with blanks to a length of four characters.

plan
: An 8-byte Db2 plan name.

retcode
: A 4-byte area in which CAF places the return code.

  This field is optional. If you do not specify *retcode*,CAF places the return code in register 15 and the reason code in register 0.

reascode
: A 4-byte area in which CAF places a reason code.

  This field is optional. If you do not specify *reascode*, CAF places the reason code in register 0. If you specify *reascode*, you must also specify *retcode*.

groupoverride
: An 8-byte area that the application provides. This field is optional. If you do not want group attach to be attempted, specify 'NOGROUP'. This string indicates that the subsystem name that is specified by *ssnm* is to be used as a Db2 subsystem name, even if *ssnm* matches a group attachment or subgroup attachment name. If you do not specify *groupoverride*, *ssnm* is used as the group attachment and subgroup attachment name if it matches a group attachment or subgroup attachment name. If you specify this parameter in any language except assembler, you must also specify *retcode* and *reascode*. In assembler language, you can omit these parameters by specifying commas as placeholders.

Recommendation: Avoid using the *groupoverride* parameter when possible, because it limits the ability to do dynamic workload routing in a Parallel Sysplex. However, you should use this parameter in a data sharing environment when you want to connect to a specific member of a data sharing group, and the subsystem name of that member is the same as the group attachment or subgroup attachment name.

Examples of CAF OPEN calls

The following table shows an OPEN call in each language.

Table 10. Examples of CAF OPEN calls

Language	Call example
Assembler	`CALL  DSNALI,(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER)`
C[1]	`fnret=dsnali(&functn[0],&ssid[0], &planname[0],&retcode, &reascode,&grpover[0]);`
COBOL	`CALL  'DSNALI' USING FUNCTN SSID PLANNAME RETCODE REASCODE GRPOVER.`
Fortran	`CALL  DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER)`
PL/I[1]	`CALL  DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER);`

Note:

- For C and PL/I applications, you must include the appropriate compiler directives, because DSNALI is an assembler language program. These compiler directives are described in the instructions for invoking CAF.

Related concepts

Implicit connections to CAF
If the CAF language interface (DSNALI) is available and you do not explicitly specify CALL DSNALI statements in your application, CAF initiates implicit CONNECT and OPEN requests to Db2. These requests are subject to the same Db2 return codes and reason codes as explicitly specified requests.

Related tasks

Invoking the call attachment facility
Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

CLOSE function for CAF

The CAF CLOSE function deallocates the plan that was created either explicitly by a call to the OPEN function or implicitly at the first SQL call. Optionally, the CLOSE function also disconnects the task, and possibly the address space, from Db2.

If you did not issue an explicit CONNECT call for the task, the CLOSE function deletes the task's connection to Db2. If no other task in the address space has an active connection to Db2, Db2 also deletes the control block structures that were created for the address space and removes the cross memory authorization.

Using the CLOSE function is optional. Consider the following rules and recommendations about when to use and not use the CLOSE function:

- Do not use the CLOSE function when your current task does not have a plan allocated.

- If you want to use a new plan, you must issue an explicit CLOSE call, followed by an OPEN call with the new plan name.
- When shutting down your application you can improve the performance of this shut down by explicitly calling the CLOSE function before the task terminates. If you omit the CLOSE call, Db2 performs an implicit CLOSE. In this case, Db2 performs the same actions when your task terminates, by using the SYNC parameter if termination is normal and the ABRT parameter if termination is abnormal.
- If Db2 terminates, issue an explicit CLOSE call for any task that did not issue a CONNECT call. This action enables CAF to reset its control blocks to allow for future connections. This CLOSE call returns the reset accomplished return code (+004) and reason code X'00C10824'. If you omit the CLOSE call in this case, when Db2 is back on line, the task's next connection request fails. You get either the message YOUR TCB DOES NOT HAVE A CONNECTION, with X'00F30018' in register 0, or the CAF error message DSNA201I or DSNA202I, depending on what your application tried to do. The task must then issue a CLOSE call before it can reconnect to Db2.
- A task that issued an explicit CONNECT call should issue a DISCONNECT call instead of a CLOSE call. This action causes CAF to reset its control blocks when Db2 terminates.

The following diagram shows the syntax for the CLOSE function.

DSNALI CLOSE function

```
>>- CALL DSNALI -- ( -- function, termop --+-----------------------------------+-- ) -><
                                           '- , -- retcode --+-----------------+-'
                                                             '- , -- reascode -'
```

Parameters point to the following areas:

function
A 12-byte area that contains the word CLOSE followed by seven blanks.

termop
A 4-byte terminate option, with one of the following values:

SYNC
Specifies that Db2 is to commit any modified data.

ABRT
Specifies that Db2 is to roll back data to the previous commit point.

retcode
A 4-byte area in which CAF is to place the return code.

This field is optional. If you do not specify *retcode*, CAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which CAF places a reason code.

This field is optional. If you do not specify *reascode*, CAF places the reason code in register 0. If you specify *reascode*, you must also specify *retcode*.

Examples of CAF CLOSE calls

The following table shows a CLOSE call in each language.

Table 11. Examples of CAF CLOSE calls

Language	Call example
Assembler	`CALL  DSNALI,(FUNCTN,TERMOP,RETCODE, REASCODE)`

Language	Call example
C[1]	`fnret=dsnali(&functn[0], &termop[0], &retcode,&reascode);`
COBOL	`CALL  'DSNALI' USING FUNCTN TERMOP RETCODE REASCODE.`
Fortran	`CALL  DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE)`
PL/I[1]	`CALL  DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE);`

Table 11. Examples of CAF CLOSE calls (continued)

Note:

- For C and PL/I applications, you must include the appropriate compiler directives, because DSNALI is an assembler language program. These compiler directives are described in the instructions for invoking CAF.

Related tasks

Invoking the call attachment facility
Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

DISCONNECT function for CAF

The CAF DISCONNECT function terminates a connection to Db2.

DISCONNECT removes the calling task's connection to Db2. If no other task in the address space has an active connection to Db2, Db2 also deletes the control block structures that were created for the address space and removes the cross memory authorization.

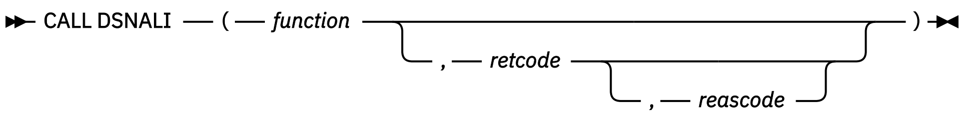
If an OPEN call is in effect, which means that a plan is allocated, when the DISCONNECT call is issued, CAF issues an implicit CLOSE with the SYNC parameter.

Using the DISCONNECT function is optional. Consider the following rules and recommendations about when to use and not use the DISCONNECT function:

- Only those tasks that explicitly issued a CONNECT call can issue a DISCONNECT call. If a CONNECT call was not used, a DISCONNECT call causes an error.
- When shutting down your application you can improve the performance of this shut down by explicitly calling the DISCONNECT function before the task terminates. If you omit the DISCONNECT call, Db2 performs an implicit DISCONNECT. In this case, Db2 performs the same actions when your task terminates.
- If Db2 terminates, any task that issued a CONNECT call must issue a DISCONNECT call to reset the CAF control blocks. The DISCONNECT function returns the reset accomplished return codes and reason codes (+004 and X'00C10824'). This action ensures that future connection requests from the task work when Db2 is back on line.
- A task that did not explicitly issue a CONNECT call must issue a CLOSE call instead of a DISCONNECT call. This action resets the CAF control blocks when Db2 terminates.

The following diagram shows the syntax for the DISCONNECT function.

DSNALI DISCONNECT function



The single parameter points to the following area:

function
A 12-byte area that contains the word DISCONNECT followed by two blanks.

retcode
A 4-byte area in which CAF places the return code.

This field is optional. If you do not specify *retcode*, CAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which CAF places a reason code.

This field is optional. If you do not specify *reascode*, CAF places the reason code in register 0. If you specify *reascode*, you must also specify *retcode*.

Examples of CAF DISCONNECT calls

The following table shows a DISCONNECT call in each language.

Table 12. Examples of CAF DISCONNECT calls

Language	Call example
Assembler	`CALL  DSNALI(,FUNCTN,RETCODE,REASCODE)`
C[1]	`fnret=dsnali(&functn[0], &retcode, &reascode);`
COBOL	`CALL  'DSNALI' USING FUNCTN RETCODE REASCODE.`
Fortran	`CALL  DSNALI(FUNCTN,RETCODE,REASCODE)`
PL/I[1]	`CALL  DSNALI(FUNCTN,RETCODE,REASCODE);`

Note:

- For C and PL/I applications, you must include the appropriate compiler directives, because DSNALI is an assembler language program. These compiler directives are described in the instructions for invoking CAF.

Related tasks

Invoking the call attachment facility

Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

TRANSLATE function for CAF

The TRANSLATE function converts a Db2 hexadecimal error reason code from a failed OPEN request into an SQL error code and printable error message text. Db2 places the information into the SQLCODE and SQLSTATE host variables or related fields of the SQLCA of the caller.

The Db2 error reason code that is converted is read from register 0. The TRANSLATE function does not change the contents of registers 0 and 15, unless the TRANSLATE request fails; in that case, register 0 is set to X'C10205' and register 15 is set to 200.

Consider the following rules and recommendations about when to use and not use the TRANSLATE function:

- You cannot call the TRANSLATE function from the Fortran language.
- The TRANSLATE function is useful only if you used an explicit CONNECT call before an OPEN request that fails. For errors that occur during SQL or IFI requests, the TRANSLATE function performs automatically.
- The TRANSLATE function can translate those codes that begin with X'00F3', but it does not translate CAF reason codes that begin with X'00C1'.

If you receive error reason code X'00F30040' (*resource unavailable*) after an OPEN request, the TRANSLATE function returns the name of the unavailable database object in the last 44 characters of the SQLERRM field.

If the TRANSLATE function does not recognize the error reason code, it returns SQLCODE -924 (SQLSTATE '58006') and places a printable copy of the original Db2 function code and the return and error reason codes in the SQLERRM field.

The following diagram shows the syntax for the TRANSLATE function.

DSNALI TRANSLATE function

```
>>-- CALL DSNALI -- ( -- function, sqlca --+------------------------------------+-- ) -><
                                           '-- , -- retcode --+---------------+-'
                                                              '-- , -- reascode -'
```

Parameters point to the following areas:

function
A 12-byte area the contains the word TRANSLATE followed by three blanks.

sqlca
The program's SQL communication area (SQLCA).

retcode
A 4-byte area in which CAF places the return code.

This field is optional. If you do not specify *retcode*, CAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which CAF places a reason code.

This field is optional. If you do not specify *reascode*, CAF places the reason code in register 0. If you specify *reascode*, you must also specify *retcode*.

Examples of CAF TRANSLATE calls

The following table shows a TRANSLATE call in each language.

Table 13. Examples of CAF TRANSLATE calls

Language	Call example
Assembler	`CALL  DSNALI,(FUNCTN,SQLCA,RETCODE, REASCODE)`
C[1]	`fnret=dsnali(&functn[0], &sqlca, &retcode, &reascode);`
COBOL	`CALL  'DSNALI' USING FUNCTN SQLCA RETCODE REASCODE.`
PL/I[1]	`CALL  DSNALI(FUNCTN,SQLCA,RETCODE, REASCODE);`

Note:

- For C and PL/I applications, you must include the appropriate compiler directives, because DSNALI is an assembler language program. These compiler directives are described in the instructions for invoking CAF.

Related tasks

Invoking the call attachment facility
Invoke the call attachment facility (CAF) when you want your application program to establish and control its own connection to Db2. Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

Turning on a CAF trace

CAF does not capture any diagnostic trace messages unless you tell it to by turning on a trace.

Procedure

Allocate a DSNTRACE data set either dynamically or by including a DSNTRACE DD statement in your JCL.

CAF writes diagnostic trace messages to that data set. The trace message numbers contain the last three digits of the reason codes.

Related concepts

Examples of invoking CAF
The call attachment facility (CAF) enables programs to communicate with Db2. If you explicitly invoke CAF in your program, you can use the CAF connection functions to control the state of the connection.

CAF return codes and reason codes

CAF provides the return codes either to the corresponding parameters that are specified in a CAF function call or, if you choose not to use those parameters, to registers 15 and 0.

When the reason code begins with X'00F3' except for X'00F30006', you can use the CAF TRANSLATE function to obtain error message text that can be printed and displayed. These reason codes are issued by the subsystem support for allied memories, a part of the Db2 subsystem support subcomponent that services all Db2 connection and work requests.

For SQL calls, CAF returns standard SQL codes in the SQLCA. CAF returns IFI return codes and reason codes in the instrumentation facility communication area (IFCA).

The following table lists the CAF return codes and reason codes.

Table 14. CAF return codes and reason codes

Return code	Reason code	Explanation
0	X'00000000'	Successful completion.

Table 14. CAF return codes and reason codes (continued)

Return code	Reason code	Explanation
4	X'00C10824'	CAF reset complete. CAF is ready to make a new connection.
8	X'00C10831'	Release level mismatch between Db2 and the CAF code.
200[1]	X'00C10201'	Received a second CONNECT request from the same TCB. The first CONNECT request could have been implicit or explicit.
200[1]	X'00C10202'	Received a second OPEN request from the same TCB. The first OPEN request could have been implicit or explicit.
200[1]	X'00C10203'	CLOSE request issued when no active OPEN request exists.
200[1]	X'00C10204'	DISCONNECT request issued when no active CONNECT request exists, or the AXSET macro was issued between the CONNECT request and the DISCONNECT request.
200[1]	X'00C10205'	TRANSLATE request issued when no connection to Db2 exists.
200[1]	X'00C10206'	Incorrect number of parameters was specified or the end-of-list bit was off.
200[1]	X'00C10207'	Unrecognized function parameter.
200[1]	X'00C10208'	Received requests to access two different Db2 subsystems from the same TCB.
204	[2]	CAF system error. Probable error in the attach or Db2.

Notes:

1. A CAF error probably caused by errors in the parameter lists from the application programs. CAF errors do not change the current state of your connection to Db2; you can continue processing with a corrected request.
2. System errors cause abends. If tracing is on, a descriptive message is written to the DSNTRACE data set just before the abend.

Sample CAF scenarios

One or more tasks can use call attachment facility (CAF) to connect to Db2. This connection can be made either implicitly or explicitly. For explicit connections, a task calls one or more of the CAF connection functions.

A single task with implicit connections

The simplest connection scenario is a single task that makes calls to Db2 without using explicit CALL DSNALI statements. The task implicitly connects to the default subsystem name and uses the default plan name.

When the task terminates, the following events occur:

- If termination was normal, any database changes are committed.
- If termination was abnormal, any database changes are rolled back.
- The active plan and all database resources are deallocated.
- The task and address space connections to Db2 are terminated.

A single task with explicit connections

The following example pseudocode illustrates a more complex scenario with a single task.

```
CONNECT
    OPEN        allocate a plan
    SQL or IFI call
    ...
    CLOSE       deallocate the current plan
    OPEN        allocate a new plan
    SQL or IFI call
    ...
    CLOSE
DISCONNECT
```

A task can have a connection to only one Db2 subsystem at any point in time. A CAF error occurs if the subsystem name in the OPEN call does not match the subsystem name in the CONNECT call. To switch to a different subsystem, the application must first disconnect from the current subsystem and then issue a connect request with a new subsystem name.

Multiple tasks

In the following scenario, multiple tasks within the address space use Db2 services. Each task must explicitly specify the same subsystem name on either the CONNECT function request or the OPEN function request. Task 1 makes no SQL or IFI calls. Its purpose is to monitor the Db2 termination and startup ECBs and to check the Db2 release level.

```
TASK 1             TASK 2          TASK 3          TASK n

CONNECT
                   OPEN            OPEN            OPEN
                   SQL             SQL             SQL
                   ...             ...             ...
                   CLOSE           CLOSE           CLOSE
                   OPEN            OPEN            OPEN
                   SQL             SQL             SQL
                   ...             ...             ...
                   CLOSE           CLOSE           CLOSE
DISCONNECT
```

Examples of invoking CAF

The call attachment facility (CAF) enables programs to communicate with Db2. If you explicitly invoke CAF in your program, you can use the CAF connection functions to control the state of the connection.

Example JCL for invoking CAF

The following sample JCL shows how to use CAF in a batch (non-TSO) environment. The DSNTRACE statement in this example is optional.

```
//jobname       JOB      z/OS_jobcard_information
//CAFJCL        EXEC     PGM=CAF_application_program
//STEPLIB       DD       DSN=application_load_library
//              DD       DSN=DB2_load_library

⋮

//SYSPRINT      DD       SYSOUT=*
//DSNTRACE      DD       SYSOUT=*
//SYSUDUMP      DD       SYSOUT=*
```

Example of assembler code that invokes CAF

The following examples show parts of a sample assembler program that uses CAF. They demonstrate the basic techniques for making CAF calls, but do not show the code and z/OS macros needed to support those calls. For example, many applications need a two-task structure so that attention-handling routines can detach connected subtasks to regain control from Db2. This structure is not shown in the following code examples. Also, these code examples assume the existence of a WRITE macro. Wherever this macro is included in the example, substitute code of your own. You must decide what you want your application to do in those situations; you probably do not want to write the error messages shown.

Example of loading and deleting the CAF language interface

The following code segment shows how an application can load entry points DSNALI and DSNHLI2 for the CAF language interface. Storing the entry points in variables LIALI and LISQL ensures that the application has to load the entry points only once. When the module is done with Db2, you should delete the entries.

```
****************************** GET LANGUAGE INTERFACE ENTRY ADDRESSES
         LOAD  EP=DSNALI          Load the CAF service request EP
         ST    R0,LIALI           Save this for CAF service requests
         LOAD  EP=DSNHLI2         Load the CAF SQL call Entry Point
         ST    R0,LISQL           Save this for SQL calls
*        .
*        .     Insert connection service requests and SQL calls here
*        .
         DELETE EP=DSNALI         Correctly maintain use count
         DELETE EP=DSNHLI2        Correctly maintain use count
```

Example of connecting to Db2 with CAF

The following example code shows how to issue explicit requests for certain actions, such as CONNECT, OPEN, CLOSE, DISCONNECT, and TRANSLATE, and uses the CHEKCODE subroutine to check the return reason codes from CAF.

```
****************************** CONNECT ********************************
         L     R15,LIALI          Get the Language Interface address
         MVC   FUNCTN,CONNECT     Get the function to call
         CALL  (15),(FUNCTN,SSID,TECB,SECB,RIBPTR),VL,MF=(E,CAFCALL)
         BAL   R14,CHEKCODE       Check the return and reason codes
         CLC   CONTROL,CONTINUE   Is everything still OK
         BNE   EXIT               If CONTROL not 'CONTINUE', stop loop
         USING R8,RIB             Prepare to access the RIB
         L     R8,RIBPTR          Access RIB to get DB2 release level
         CLC   RIBREL,RIBR999     DB2 V10 or later?
         BE    USERELX            If RIBREL = '999', use RIBRELX
         WRITE 'The current DB2 release level is' RIBREL
         B     OPEN               Continue with signon
         USERELX  WRITE 'The current DB2 release level is' RIBRELX

****************************** OPEN ***********************************
OPEN     L     R15,LIALI          Get the Language Interface address
         MVC   FUNCTN,OPEN        Get the function to call
         CALL  (15),(FUNCTN,SSID,PLAN),VL,MF=(E,CAFCALL)
         BAL   R14,CHEKCODE       Check the return and reason codes

****************************** SQL ************************************
*              Insert your SQL calls here.  The DB2 Precompiler
*              generates calls to entry point DSNHLI.  You should
*              specify the precompiler option ATTACH(CAF), or code
*              a dummy entry point named DSNHLI to intercept
*              all SQL calls. A dummy DSNHLI is shown below.
****************************** CLOSE **********************************
         CLC   CONTROL,CONTINUE   Is everything still OK?
         BNE   EXIT               If CONTROL not 'CONTINUE', shut down
         MVC   TRMOP,ABRT         Assume termination with ABRT parameter
         L     R4,SQLCODE         Put the SQLCODE into a register
         C     R4,CODE0           Examine the SQLCODE
         BZ    SYNCTERM           If zero, then CLOSE with SYNC parameter
         C     R4,CODE100         See if SQLCODE was 100
         BNE   DISC               If not 100, CLOSE with ABRT parameter
SYNCTERM MVC   TRMOP,SYNC         Good code, terminate with SYNC parameter
DISC     DS    0H                 Now build the CAF parmlist
         L     R15,LIALI          Get the Language Interface address
         MVC   FUNCTN,CLOSE       Get the function to call
         CALL  (15),(FUNCTN,TRMOP),VL,MF=(E,CAFCALL)
         BAL   R14,CHEKCODE       Check the return and reason codes

****************************** DISCONNECT *****************************
         CLC   CONTROL,CONTINUE   Is everything still OK
         BNE   EXIT               If CONTROL not 'CONTINUE', stop loop
         L     R15,LIALI          Get the Language Interface address
         MVC   FUNCTN,DISCON      Get the function to call
         CALL  (15),(FUNCTN),VL,MF=(E,CAFCALL)
         BAL   R14,CHEKCODE       Check the return and reason codes
```

This example code does not show a task that waits on the Db2 termination ECB. If you want such a task, you can code it by using the z/OS WAIT macro to monitor the ECB. You probably want this task to detach the sample code if the termination ECB is posted. That task can also wait on the Db2 startup ECB. This sample waits on the startup ECB at its own task level.

This example code assumes that the variables in the following table are already set:

Table 15. Variables that preceding example assembler code assumes are set

Variable	Usage
LIALI	The entry point that handles Db2 connection service requests.
LISQL	The entry point that handles SQL calls.
SSID	The Db2 subsystem identifier.
TECB	The address of the Db2 termination ECB.
SECB	The address of the Db2 startup ECB.
RIBPTR	A fullword that CAF sets to contain the RIB address.
PLAN	The plan name to use in the OPEN call.
CONTROL	This variable is used to shut down processing because of unsatisfactory return or reason codes. The CHECKCODE subroutine sets this value.
CAFCALL	List-form parameter area for the CALL macro.

Example of checking return codes and reason codes when using CAF

The following example code illustrates a way to check the return codes and the Db2 termination ECB after each connection service request and SQL call. The routine sets the variable CONTROL to control further processing within the module.

```
***********************************************************************
* CHEKCODE PSEUDOCODE                                                 *
***********************************************************************
*IF TECB is POSTed with the ABTERM or FORCE codes
*  THEN
*    CONTROL = 'SHUTDOWN'
*    WRITE 'DB2 found FORCE or ABTERM, shutting down'
*  ELSE                          /* Termination ECB was not POSTed  */
*    SELECT (RETCODE)            /* Look at the return code         */
*      WHEN (0) ;                /* Do nothing; everything is OK    */
*      WHEN (4) ;                /* Warning                         */
*        SELECT (REASCODE)       /* Look at the reason code         */
*          WHEN ('00C10824'X)    /* Ready for another CAF call      */
*            CONTROL = 'RESTART'  /* Start over, from the top        */
*          OTHERWISE
*            WRITE 'Found unexpected R0 when R15 was 4'
*            CONTROL = 'SHUTDOWN'
*        END INNER-SELECT
*      WHEN (8,12)               /* Connection failure              */
*        SELECT (REASCODE)       /* Look at the reason code         */
*          WHEN ('00C10831'X)    /* DB2 / CAF release level mismatch*/
*            WRITE 'Found a mismatch between DB2 and CAF release levels'
*          WHEN ('00F30002'X,    /* These mean that DB2 is down but */
*                '00F30012'X)    /* will POST SECB when up again    */
*            DO
*              WRITE 'DB2 is unavailable. I'll tell you when it is up.'
*              WAIT SECB         /* Wait for DB2 to come up         */
*              WRITE 'DB2 is now available.'
*            END
*          /**********************************************************/
*          /* Insert tests for other DB2 connection failures here.   */
*          /* CAF Externals Specification lists other codes you can  */
*          /* receive.  Handle them in whatever way is appropriate   */
*          /* for your application.                                  */
```

```
*          /*********************************************************/
*          OTHERWISE              /* Found a code we're not ready for*/
*            WRITE 'Warning: DB2 connection failure. Cause unknown'
*            CALL DSNALI ('TRANSLATE',SQLCA) /* Fill in SQLCA       */
*            WRITE SQLCODE and SQLERRM
*        END INNER-SELECT
*      WHEN (200)
*        WRITE 'CAF found user error. See DSNTRACE data set'
*      WHEN (204)
*        WRITE 'CAF system error. See DSNTRACE data set'
*      OTHERWISE
*        CONTROL = 'SHUTDOWN'
*        WRITE 'Got an unrecognized return code'
*    END MAIN SELECT
*    IF (RETCODE > 4) THEN         /* Was there a connection problem?*/
*      CONTROL = 'SHUTDOWN'
* END CHEKCODE
```

```
***********************************************************************
* Subroutine CHEKCODE checks return codes from DB2 and Call Attach.
* When CHEKCODE receives control, R13 should point to the caller's
* save area.
***********************************************************************
CHEKCODE DS    0H
         STM   R14,R12,12(R13)    Prolog
         ST    R15,RETCODE        Save the return code
         ST    R0,REASCODE        Save the reason code
         LA    R15,SAVEAREA       Get save area address
         ST    R13,4(,R15)        Chain the save areas
         ST    R15,8(,R13)        Chain the save areas
         LR    R13,R15            Put save area address in R13
*        ********************* HUNT FOR FORCE OR ABTERM ***************
         TM    TECB,POSTBIT       See if TECB was POSTed
         BZ    DOCHECKS           Branch if TECB was not POSTed
         CLC   TECBCODE(3),QUIESCE   Is this "STOP DB2 MODE=FORCE"
         BE    DOCHECKS           If not QUIESCE, was FORCE or ABTERM
         MVC   CONTROL,SHUTDOWN   Shutdown
         WRITE 'Found found FORCE or ABTERM, shutting down'
         B     ENDCCODE           Go to the end of CHEKCODE
DOCHECKS DS    0H                 Examine RETCODE and REASCODE
*        ********************* HUNT FOR 0 *****************************
         CLC   RETCODE,ZERO       Was it a zero?
         BE    ENDCCODE           Nothing to do in CHEKCODE for zero
*        ********************* HUNT FOR 4 *****************************
         CLC   RETCODE,FOUR       Was it a 4?
         BNE   HUNT8              If not a 4, hunt eights
         CLC   REASCODE,C10831    Was it a release level mismatch?
         BNE   HUNT824            Branch if not an 831
         WRITE 'Found a mismatch between DB2 and CAF release levels'
         B     ENDCCODE           We are done. Go to end of CHEKCODE
HUNT824  DS    0H                 Now look for 'CAF reset' reason code
         CLC   REASCODE,C10824    Was it 4? Are we ready to restart?
         BNE   UNRECOG            If not 824, got unknown code
         WRITE 'CAF is now ready for more input'
         MVC   CONTROL,RESTART    Indicate that we should re-CONNECT
         B     ENDCCODE           We are done. Go to end of CHEKCODE
UNRECOG  DS    0H
         WRITE 'Got RETCODE = 4 and an unrecognized reason code'
         MVC   CONTROL,SHUTDOWN   Shutdown, serious problem
         B     ENDCCODE           We are done. Go to end of CHEKCODE
*        ********************* HUNT FOR 8 *****************************
HUNT8    DS    0H
         CLC   RETCODE,EIGHT      Hunt return code of 8
         BE    GOT8OR12
         CLC   RETCODE,TWELVE     Hunt return code of 12
         BNE   HUNT200
GOT8OR12 DS    0H                 Found return code of 8 or 12
         WRITE 'Found RETCODE of 8 or 12'
         CLC   REASCODE,F30002    Hunt for X'00F30002'
         BE    DB2DOWN
```

```
         CLC   REASCODE,F30012    Hunt for X'00F30012'
         BE    DB2DOWN
         WRITE 'DB2 connection failure with an unrecognized REASCODE'
         CLC   SQLCODE,ZERO       See if we need TRANSLATE
         BNE   A4TRANS            If not blank, skip TRANSLATE
*        ********************* TRANSLATE unrecognized RETCODEs ********
         WRITE 'SQLCODE 0 but R15 not, so TRANSLATE to get SQLCODE'
         L     R15,LIALI          Get the Language Interface address
         CALL  (15),(TRANSLAT,SQLCA),VL,MF=(E,CAFCALL)
```

```
         C     R0,C10205           Did the TRANSLATE work?
         BNE   A4TRANS             If not C10205, SQLERRM now filled in
         WRITE 'Not able to TRANSLATE the connection failure'
         B     ENDCCODE            Go to end of CHEKCODE
A4TRANS  DS    0H                  SQLERRM must be filled in to get here
*        Note: your code should probably remove the X'FF'
*        separators and format the SQLERRM feedback area.
*        Alternatively, use DB2 Sample Application DSNTIAR
*        to format a message.
         WRITE 'SQLERRM is:' SQLERRM
         B     ENDCCODE            We are done. Go to end of CHEKCODE
DB2DOWN  DS    0H                  Hunt return code of 200
         WRITE 'DB2 is down and I will tell you when it comes up'
         WAIT  ECB=SECB            Wait for DB2 to come up
         WRITE 'DB2 is now available'
         MVC   CONTROL,RESTART     Indicate that we should re-CONNECT
         B     ENDCCODE
*        ********************* HUNT FOR 200 **************************
HUNT200  DS    0H                  Hunt return code of 200
         CLC   RETCODE,NUM200      Hunt 200
         BNE   HUNT204
         WRITE 'CAF found user error, see DSNTRACE data set'
         B     ENDCCODE            We are done. Go to end of CHEKCODE
*        ********************* HUNT FOR 204 **************************
HUNT204  DS    0H                  Hunt return code of 204
         CLC   RETCODE,NUM204      Hunt 204
         BNE   WASSAT              If not 204, got strange code
         WRITE 'CAF found system error, see DSNTRACE data set'
         B     ENDCCODE            We are done. Go to end of CHEKCODE
*        ********************* UNRECOGNIZED RETCODE *******************
WASSAT   DS    0H
         WRITE 'Got an unrecognized RETCODE'
         MVC   CONTROL,SHUTDOWN    Shutdown
         BE    ENDCCODE            We are done. Go to end of CHEKCODE
ENDCCODE DS    0H                  Should we shut down?
         L     R4,RETCODE          Get a copy of the RETCODE
         C     R4,FOUR             Have a look at the RETCODE
         BNH   BYEBYE              If RETCODE <= 4 then leave CHEKCODE
         MVC   CONTROL,SHUTDOWN    Shutdown
BYEBYE   DS    0H                  Wrap up and leave CHEKCODE
         L     R13,4(,R13)         Point to caller's save area
         RETURN (14,12)            Return to the caller
```

Example of invoking CAF when you do not specify the precompiler option ATTACH(CAF)

Each of the four Db2 attachment facilities contains an entry point named DSNHLI. When you use CAF but do not specify the precompiler option ATTACH(CAF), SQL statements result in BALR instructions to DSNHLI in your program. To find the correct DSNHLI entry point without including DSNALI in your load module, code a subroutine with entry point DSNHLI that passes control to entry point DSNHLI2 in the DSNALI module. DSNHLI2 is unique to DSNALI and is at the same location in DSNALI as DSNHLI. DSNALI uses 31-bit addressing. If the application that calls this intermediate subroutine uses 24-bit addressing, this subroutine should account for the difference.

In the following example, LISQL is addressable because the calling CSECT used the same register 12 as CSECT DSNHLI. Your application must also establish addressability to LISQL.

```
***********************************************************************
* Subroutine DSNHLI intercepts calls to LI EP=DSNHLI
***********************************************************************
         DS    0D
DSNHLI   CSECT                     Begin CSECT
         STM   R14,R12,12(R13)     Prologue
         LA    R15,SAVEHLI         Get save area address
         ST    R13,4(,R15)         Chain the save areas
         ST    R15,8(,R13)         Chain the save areas
         LR    R13,R15             Put save area address in R13
         L     R15,LISQL           Get the address of real DSNHLI
         BASSM R14,R15             Branch to DSNALI to do an SQL call
*                                  DSNALI is in 31-bit mode, so use
*                                  BASSM to assure that the addressing
*                                  mode is preserved.
         L     R13,4(,R13)         Restore R13 (caller's save area addr)
         L     R14,12(,R13)        Restore R14 (return address)
         RETURN (1,12)             Restore R1-12, NOT R0 and R15 (codes)
```

Example of variable declarations when using CAF

The following example code shows declarations for some of the variables that were used in the previous subroutines.

```
****************************** VARIABLES ******************************
SECB     DS    F                  DB2 Startup ECB
TECB     DS    F                  DB2 Termination ECB
LIALI    DS    F                  DSNALI Entry Point address
LISQL    DS    F                  DSNHLI2 Entry Point address
SSID     DS    CL4                DB2 Subsystem ID.  CONNECT parameter
PLAN     DS    CL8                DB2 Plan name.  OPEN parameter
TRMOP    DS    CL4                CLOSE termination option (SYNC|ABRT)
FUNCTN   DS    CL12               CAF function to be called
RIBPTR   DS    F                  DB2 puts Release Info Block addr here
RETCODE  DS    F                  Chekcode saves R15 here
REASCODE DS    F                  Chekcode saves R0 here
CONTROL  DS    CL8                GO, SHUTDOWN, or RESTART
SAVEAREA DS  18F                  Save area for CHEKCODE
****************************** CONSTANTS ******************************
SHUTDOWN DC    CL8'SHUTDOWN'      CONTROL value: Shutdown execution
RESTART  DC    CL8'RESTART '      CONTROL value: Restart execution
CONTINUE DC    CL8'CONTINUE'      CONTROL value: Everything OK, cont
CODE0    DC    F'0'               SQLCODE of 0
CODE100  DC    F'100'             SQLCODE of 100
QUIESCE  DC    XL3'000008'        TECB postcode: STOP DB2 MODE=QUIESCE
CONNECT  DC    CL12'CONNECT     ' Name of a CAF service. Must be CL12!
OPEN     DC    CL12'OPEN        ' Name of a CAF service. Must be CL12!
CLOSE    DC    CL12'CLOSE       ' Name of a CAF service. Must be CL12!
DISCON   DC    CL12'DISCONNECT  ' Name of a CAF service. Must be CL12!
TRANSLAT DC    CL12'TRANSLATE   ' Name of a CAF service. Must be CL12!
SYNC     DC    CL4'SYNC'          Termination option (COMMIT)
ABRT     DC    CL4'ABRT'          Termination option (ROLLBACK)
****************************** RETURN CODES (R15) FROM CALL ATTACH ****
ZERO     DC    F'0'               0
FOUR     DC    F'4'               4
EIGHT    DC    F'8'               8
TWELVE   DC    F'12'              12  (Call Attach return code in R15)
NUM200   DC    F'200'             200 (User error)
NUM204   DC    F'204'             204 (Call Attach system error)
****************************** REASON CODES (R00) FROM CALL ATTACH ****
C10205   DC    XL4'00C10205'      Call attach could not TRANSLATE
C10831   DC    XL4'00C10831'      Call attach found a release mismatch
C10824   DC    XL4'00C10824'      Call attach ready for more input
F30002   DC    XL4'00F30002'      DB2 subsystem not up
F30011   DC    XL4'00F30011'      DB2 subsystem not up
F30012   DC    XL4'00F30012'      DB2 subsystem not up
F30025   DC    XL4'00F30025'      DB2 is stopping (REASCODE)
*
*        Insert more codes here as necessary for your application
*
****************************** SQLCA and RIB **************************
    EXEC SQL INCLUDE SQLCA
         DSNDRIB                  Get the DB2 Release Information Block
****************************** CALL macro parm list *******************
CAFCALL  CALL  ,(*,*,*,*,*,*,*,*,*),VL,MF=L
```

Invoking the Resource Recovery Services attachment facility

The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

Before you begin

Before you invoke RRSAF, perform the following actions:

- Ensure that the RRSAF language interface load module, DSNRLI, is available.
- Ensure that your application satisfies the requirements for programs that access RRSAF.
- Ensure that your application satisfies the general environment characteristics for connecting to Db2.
- Ensure that you are familiar with the following z/OS concepts and facilities:
  - The CALL macro and standard module linkage conventions

- Program addressing and residency options (AMODE and RMODE)
- Creating and controlling tasks; multitasking
- Functional recovery facilities such as ESTAE, ESTAI, and FRRs
- Synchronization techniques such as WAIT/POST
- z/OS RRS functions, such as SRRCMIT and SRRBACK

About this task

Applications that use RRSAF can be written in assembler language, C, COBOL, Fortran, and PL/I. When choosing a language to code your application in, consider the following restrictions:

- If you use z/OS macros (ATTACH, WAIT, POST, and so on), choose a programming language that supports them.
- The RRSAF TRANSLATE function is not available in Fortran. To use this function, code it in a routine that is written in another language, and then call that routine from Fortran.

Procedure

To invoke RRSAF:

1. Perform one of the following actions:
   - Explicitly invoke RRSAF by including in your program CALL DSNRLI statements with the appropriate options.

     The first option is an RRSAF connection function, which describes the action that you want RRSAF to take. The effect of any function depends in part on what functions the program has already performed.

     To code RRSAF functions in C, COBOL, Fortran, or PL/I, follow the individual language's rules for making calls to assembler language routines. Specify the return code and reason code parameters in the parameter list for each RRSAF call.

     Requirement: For C, C++, and PL/I applications, you must also include in your program the compiler directives that are listed in the following table, because DSNRLI is an assembler language program.

     Table 16. Compiler directives to include in C, C++, and PL/I applications that contain CALL DSNRLI statements

Language	Compiler directive to include
C	`#pragma linkage(dsnrli, OS)`
C++	`extern "OS" { int DSNRLI( char * functn, ...); }`
PL/I	`DCL DSNRLI ENTRY OPTIONS(ASM,INTER,RETCODE);`

   - Implicitly invoke RRSAF by including SQL statements or IFI calls in your program just as you would in any program. The RRSAF facility establishes the connection to Db2 with the default values for the subsystem name, plan name and authorization ID.

     Restriction: If your program can make its first SQL call from different modules with different DBRMs, you cannot use a default plan name and thus, you cannot implicitly invoke RRSAF. Instead, you must explicitly invoke RRSAF by calling the CREATE THREAD function.

     Requirement: If your application includes both SQL and IFI calls, you must issue at least one SQL call before you issue any IFI calls. This action ensures that your application uses the correct plan.

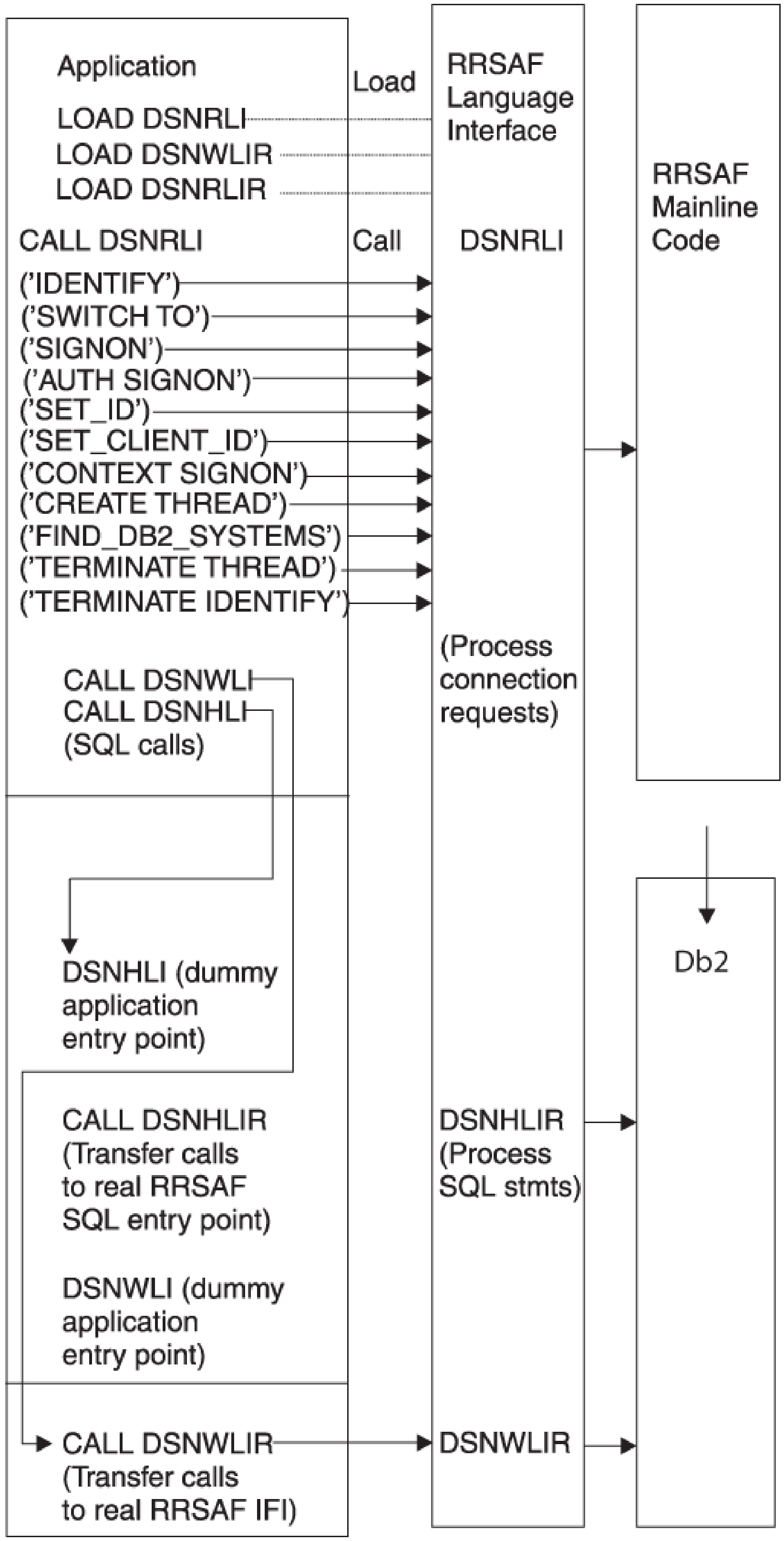
2. If you implicitly invoked RRSAF, determine if the implicit connection was successful by examining the return code and reason code immediately after the first executable SQL statement within the application program. Your program can check these codes by performing one of the following actions:

   - Examine registers 0 and 15 directly.
   - Examine the SQLCA, and if the SQLCODE is -981, obtain the return and reason code from the message text. The return code is the first token, and the reason code is the second token.

   If the implicit connection is successful, the application can examine the SQLCODE for the first, and subsequent, SQL statements.

Example of an RRSAF configuration

The following figure shows an conceptual example of invoking and using RRSAF.



Resource Recovery Services attachment facility

An attachment facility enables programs to communicate with Db2. The Resource Recovery Services attachment facility (RRSAF) provides such a connection for programs that run in z/OS batch, TSO foreground, and TSO background. The RRSAF is an alternative to CAF and has more functionality.

An application program using RRSAF can perform the following actions:

- Use Db2 to process SQL statements, commands, or instrumentation facility interface (IFI) calls.
- Coordinate Db2 updates with updates made by all other resource managers that also use z/OS RRS in an z/OS system.
- Use the z/OS System Authorization Facility and an external security product, such as RACF, to sign on to Db2 with the authorization ID of a user.
- Sign on to Db2 using a new authorization ID and an existing connection and plan.
- Access Db2 from multiple z/OS tasks in an address space.
- Switch a Db2 thread among z/OS tasks within a single address space.
- Access the Db2 IFI.
- Run with or without the TSO terminal monitor program (TMP).
- Run without being a subtask of the DSN command processor (or of any Db2 code).
- Run above or below the 16-MB line.
- Establish an explicit connection to Db2, through a call interface, with control over the exact state of the connection.
- Establish an implicit connection to Db2 (with a default subsystem identifier and a default plan name) by using SQL statements or IFI calls without first calling RRSAF.
- Supply event control blocks (ECBs), for Db2 to post, that signal start-up or termination.
- Intercept return codes, reason codes, and abend codes from Db2 and translate them into messages as required.

RRSAF uses z/OS Transaction Management and Recoverable Resource Manager Services (z/OS RRS).

Any task in an address space can establish a connection to Db2 through RRSAF. Each task control block (TCB) can have only one connection to Db2. A Db2 service request that is issued by a program that runs under a given task is associated with that task's connection to Db2. The service request operates independently of any Db2 activity under any other task.

Each connected task can run a plan. Tasks within a single address space can specify the same plan, but each instance of a plan runs independently from the others. A task can terminate its plan and run a different plan without completely breaking its connection to Db2.

RRSAF does not generate task structures.

When you design your application, consider that using multiple simultaneous connections can increase the possibility of deadlocks and Db2 resource contention.

Restriction: RRSAF does not provide attention processing exits or functional recovery routines. You can provide whatever attention handling and functional recovery your application needs, but you must use ESTAE/ESTAI type recovery routines only.

A tracing facility provides diagnostic messages that help you debug programs and diagnose errors in the RRSAF code. The trace information is available only in a SYSABEND or SYSUDUMP dump.

To commit work in RRSAF applications, use the CPIC SRRCMIT function or the Db2 COMMIT statement. To roll back work, use the CPIC SRRBACK function or the Db2 ROLLBACK statement.

Use the following guidelines to decide whether to use the Db2 statements or the CPIC functions for commit and rollback operations:

- Use Db2 COMMIT and ROLLBACK statements when all of the following conditions are true:
  - The only recoverable resource that is accessed by your application is Db2 data that is managed by a single Db2 instance.

    Db2 COMMIT and ROLLBACK statements fail if your RRSAF application accesses recoverable resources other than Db2 data that is managed by a single Db2 instance.
  - The address space from which syncpoint processing is initiated is the same as the address space that is connected to Db2.

- If your application accesses other recoverable resources, or syncpoint processing and Db2 access are initiated from different address spaces, use SRRCMIT and SRRBACK.

Related reference

COMMIT (Db2 SQL)

ROLLBACK (Db2 SQL)

Related information

Using Protected Resources (MVS Programming: Callable Services for High-Level Languages)

Properties of RRSAF connections

RRSAF enables programs to communicate with Db2 to process SQL statements, commands, or IFI calls.

Restriction: Do not mix RRSAF connections with other connection types in a single address space. The first connection that is made from an address space to Db2 determines the type of connection allowed.

The connection that RRSAF makes with Db2 has the basic properties that are listed in the following table.

Table 17. Properties of RRSAF connections

Property	Value	Comments
Connection name	RRSAF	You can use the DISPLAY THREAD command to list RRSAF applications that have the connection name RRSAF.
Connection type	RRSAF	None.

Property	Value	Comments
Authorization ID	Authorization IDs that are associated with each Db2 connection	A connection must have a primary ID and can have one or more secondary IDs. Those identifiers are used for the following purposes: • Validating access to Db2 • Checking privileges on Db2 objects • Assigning ownership of Db2 objects • Identifying the user of a connection for audit, performance, and accounting traces. RRSAF relies on the z/OS System Authorization Facility (SAF) and a security product, such as RACF, to verify and authorize the authorization IDs. An application that connects to Db2 through RRSAF must pass those identifiers to SAF for verification and authorization checking. RRSAF retrieves the identifiers from SAF. A location can provide an authorization exit routine for a Db2 connection to change the authorization IDs and to indicate whether the connection is allowed. The actual values that are assigned to the primary and secondary authorization IDs can differ from the values that are provided by a SIGNON or AUTH SIGNON request. A site's Db2 signon exit routine can access the primary and secondary authorization IDs and can modify the IDs to satisfy the site's security requirements. The exit routine can also indicate whether the signon request should be accepted.

Table 17. Properties of RRSAF connections (continued)

Table 17. Properties of RRSAF connections (continued)

Property	Value	Comments
Scope	RRSAF processes connections as if each task is entirely isolated. When a task requests a function, RRSAF passes the function to Db2, regardless of the connection status of other tasks in the address space. However, the application program and the Db2 subsystem have access to the connection status of multiple tasks in an address space.	None.

If an application that is connected to Db2 through RRSAF terminates normally before the TERMINATE THREAD or TERMINATE IDENTIFY functions deallocate the plan, RRS commits any changes made after the last commit point. If the application terminates abnormally before the TERMINATE THREAD or TERMINATE IDENTIFY functions deallocate the plan, z/OS RRS rolls back any changes made after the last commit point. In either case, Db2 deallocates the plan, if necessary, and terminates the application's connection.

If Db2 abends while an application is running, Db2 rolls back changes to the last commit point. If Db2 terminates while processing a commit request, Db2 either commits or rolls back any changes at the next restart. The action taken depends on the state of the commit request when Db2 terminates.

Making the RRSAF language interface (DSNRLI) available

Before you can invoke the Resource Recovery Services attachment facility (RRSAF), you must first make available the RRSAF language interface load module, DSNRLI.

About this task

Part of RRSAF is a Db2 load module, DSNRLI, which is also known as the RRSAF language interface module. DSNRLI has the alias names DSNHLIR and DSNWLIR. The module has five entry points: DSNRLI, DSNHLI, DSNHLIR, DSNWLI, and DSNWLIR. These entry points serve the following functions:

- Entry point DSNRLI handles explicit Db2 connection service requests.
- DSNHLI and DSNHLIR handle SQL calls. Use DSNHLI if your application program link-edits RRSAF. Use DSNHLIR if your application program loads RRSAF.
- DSNWLI and DSNWLIR handle IFI calls. Use DSNWLI if your application program link-edits RRSAF. Use DSNWLIR if your application program loads RRSAF.

Procedure

To make DSNRLI available:

1. Decide which of the following methods you want to use to make DSNRLI available:
   - Explicitly issuing LOAD requests when your program runs.

     By explicitly loading the DSNRLI module, you can isolate the maintenance of your application from future IBM maintenance to the language interface. If the language interface changes, the change will probably not affect your load module.
   - Including the DSNRLI module in your load module when you link-edit your program.

     A disadvantage of link-editing DSNRLI into your load module is that if IBM makes a change to DSNRLI, you must link-edit your program again.

Alternatively, if using explicit connections via CALL DSNALI, you can link-edit your program with DSNULI, the Universal Language Interface.

2. Depending on the method that you chose in step 1, perform one of the following actions:
   - **If you want to explicitly issue LOAD requests when your program runs:**

     In your program, issue z/OS LOAD service requests for entry points DSNRLI and DSNHLIR. If you use IFI services, you must also load DSNWLIR. Save the entry point address that LOAD returns and use it in the CALL macro.

     Indicate to Db2 which entry point to use in one of the following two ways:

     – Specify the precompiler option ATTACH(RRSAF).

       This option causes Db2 to generate calls that specify entry point DSNHLIR.

       Restriction: You cannot use this option if your application is written in Fortran.

     – Code a dummy entry point named DSNHLI within your load module.

       If you do not specify the precompiler option ATTACH, the Db2 precompiler generates calls to entry point DSNHLI for each SQL request. The precompiler does not know about and is independent of the different Db2 attachment facilities. When the calls that are generated by the Db2 precompiler pass control to DSNHLI, your code that corresponds to the dummy entry point must preserve the option list that is passed in register 1 and call DSNHLIR with the same option list.
   - **If you want to include the DSNRLI module in your load module when you link-edit your program:**

     Include DSNRLI in your load module during a link-edit step. For example, you can use a linkage editor control statement that is similar to the following statement in your JCL:

     ```
     INCLUDE DB2LIB(DSNRLI).
     ```

     By coding this statement, you avoid inadvertently picking up the wrong language interface module.

     When you include the DSNRLI module during the link-edit, do not include a dummy DSNHLI entry point in your program or specify the precompiler option ATTACH. Module DSNRLI contains an entry point for DSNHLI, which is identical to DSNHLIR, and an entry point for DSNWLI, which is identical to DSNWLIR.

Related concepts

Program examples for RRSAF
The Resource Recovery Services attachment facility (RRSAF) enables programs to communicate with Db2. You can use RRSAF as an alternative to CAF.

"Universal language interface (DSNULI)" on page 117
The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

Related tasks

Making the CAF language interface (DSNALI) available
Before you can invoke the call attachment facility (CAF), you must first make DSNALI available.

Link-editing an application with DSNULI
To create a single load module that can be used in more than one attachment environment, you can link-edit your program or stored procedure with the Universal Language Interface module (DSNULI) instead of with one of the environment-specific language interface modules (DSNELI, DSNALI, DSNRLI, or DSNCLI).

Requirements for programs that use RRSAF

The Resource Recovery Services attachment facility (RRSAF) enables programs to communicate with Db2. Before you invoke RRSAF in your program, ensure that your program satisfies any requirements for using RRSAF.

When you write programs that use RRSAF, ensure that they meet the following requirements:

- The program accounts for the size of the RRSAF code. The RRSAF code requires about 10 KB of virtual storage per address space and an additional 10 KB for each TCB that uses RRSAF.
- If your local environment intercepts and replaces the z/OS LOAD SVC that RRSAF uses, you must ensure that your version of LOAD manages the load list element (LLE) and contents directory entry (CDE) chains like the standard z/OS LOAD macro. RRSAF uses z/OS SVC LOAD to load a module as part of the initialization after your first service request. The module is loaded into fetch-protected storage that has the job-step protection key.

You can prepare application programs to run in RRSAF similar to how you prepare applications to run in other environments, such as CICS, IMS, and TSO. You can prepare an RRSAF application either in the batch environment or by using the Db2 program preparation process. You can use the program preparation system either through DB2I or through the DSNH CLIST.

Related tasks

Preparing an application to run on Db2 for z/OS
To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

How RRSAF modifies the content of registers

If you do not specify the return code and reason code parameters in your RRSAF function calls or ifyou invoke RRSAF implicitly, RRSAF puts a return code in register 15 and a reason code in register 0. RRSAF preserves the contents of registers 2 through 14.

If you specify the return code and reason code parameters, RRSAF places the return code in register 15 and in the return code parameter to accommodate high-level languages that support special return code processing.

The following table summarizes the register conventions for RRSAF calls.

Table 18. Register conventions for RRSAF calls

Register	Usage
R1	Parameter list pointer
R13	Address of caller's save area
R14	Caller's return address
R15	RRSAF entry point address

Implicit connections to RRSAF

Resource Recovery Services attachment facility (RRSAF) establishes an implicit connection to Db2 under certain situations. The connection is established if the following are true: the RRSAF language interface load module (DSNRLI) is available, you do not explicitly specify the IDENTIFY function in a CALL DSNRLI statement in your program, and the application includes SQL statements or IFI calls.

An implicit connection causes RRSAF to initiate implicit IDENTIFY and CREATE THREAD requests to Db2. These requests are subject to the same Db2 return codes and reason codes as explicitly specified requests.

Implicit connections use the following defaults:

Subsystem name

The default name that is specified in the module DSNHDECP. RRSAF uses the installation default DSNHDECP, unless your own DSNHDECP module is in a library in a STEPLIB statement of the JOBLIB concatenation or in the link list. In a data sharing group, the default subsystem name is the group attachment name.

Be certain that you know what the default name is and that it names the specific Db2 subsystem that you want to use.

Plan name
: The member name of the database request module (DBRM) that Db2 produced when you precompiled the source program that contains the first SQL call.

Authorization ID
: The 7-byte user ID that is associated with the address space, unless an authorized function has built an Accessor Environment Element (ACEE) for the address space. If an authorized function has built an ACEE, Db2 passes the 8-byte user ID from the ACEE.

For an implicit connection request, your application should not explicitly specify either the IDENTIFY function or the CREATE THREAD function. Your application can execute other explicit RRSAF calls after the implicit connection is made. An implicit connection does not perform any SIGNON processing. Your application can execute the SIGNON function at any point of consistency. To terminate an implicit connection, you must use the proper function calls.

For implicit connection requests, register 15 contains the return code, and register 0 contains the reason code. The return code and reason code are also in the message text for SQLCODE -981.

Related concepts

Summary of RRSAF behavior
The effect of any Resource Recovery Services attachment facility (RRSAF) function depends in part on what functions the program has already run. You should plan the RRSAF function calls that your program makes to avoid any errors and major structural problems in your application.

CALL DSNRLI statement parameter list

The CALL DSNRLI statement explicitly invokes RRSAF. When you include CALL DSNRLI statements in your program, you must specify all parameters that precede the return code parameter.

In CALL DSNRLI statements, you cannot omit any of parameters that come before the return code parameter by coding zeros or blanks. No defaults exist for those parameters for explicit connection requests. Defaults are provided for only implicit connections. All parameters starting with the return code parameter are optional.

When you want to use the default value for a parameter but specify subsequent parameters, code the CALL DSNRLI statement as follows:

- For C-language, when you code CALL DSNRLI statements in C, you need to specify the address of every parameter, using the "address of" operator (&), and not the parameter itself. For example, to pass the pklistptr parameter on the "CREATE THREAD" specify the address of the 4-byte pointer to the structure (&pklistptr):

  ```
  fnret=dsnrli(&crthrdfn[0], &plan[0], &collid[0], &reuse[0],
     &retcode, &reascode, &pklistptr);
  ```

- For all languages except assembler language, code zero for that parameter in the CALL DSNRLI statement. For example, suppose that you are coding an IDENTIFY call in a COBOL program, and you want to specify all parameters except the return code parameter. You can write a statement similar to the following statement:

  ```
  CALL 'DSNRLI' USING IDFYFN SSNM RIBPTR EIBPTR TERMECB STARTECB
     BY CONTENT ZERO BY REFERENCE REASCODE.
  ```

- For assembler language, code a comma for that parameter in the CALL DSNRLI statement. For example, suppose that you are coding an IDENTIFY call, and you want to specify all parameters except the return code parameter. You can write a statement similar to the following statement:

  ```
  CALL  DSNRLI,(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB,,REASCODE)
  ```

For assembler programs that invoke RRSAF, use a standard parameter list for an z/OS CALL. Register 1 must contain the address of a list of pointers to the parameters. Each pointer is a 4-byte address. The last address must contain the value 1 in the high-order bit.

Summary of RRSAF behavior

The effect of any Resource Recovery Services attachment facility (RRSAF) function depends in part on what functions the program has already run. You should plan the RRSAF function calls that your program makes to avoid any errors and major structural problems in your application.

The following tables summarize RRSAF behavior after various inputs from application programs. The contents of each table cell indicate the result of calling the function in the first column for that row followed by the function in the current column heading. For example, if you issue TERMINATE THREAD and then IDENTIFY, RRSAF returns reason code X'00C12201'. Use these tables to understand the order in which your application must issue RRSAF calls, SQL statements, and IFI requests.

The RRSAF FIND_DB2_SYSTEMS function is omitted from these tables, because it does not affect the operation of any of the other functions

The following table summarizes RRSAF behavior when the next call is to the IDENTIFY function, the SWITCH TO function, the SIGNON function, or the CREATE THREAD function.

Table 19. Effect of call order when next call is IDENTIFY, SWITCH TO, SIGNON, or CREATE THREAD

	Next function			
Previous function	**IDENTIFY**	**SWITCH TO**	**SIGNON, AUTH SIGNON, or CONTEXT SIGNON**	**CREATE THREAD**
Empty: first call	IDENTIFY	X'00C12205'[1]	X'00C12204'[1]	X'00C12204'[1]
IDENTIFY	X'00F30049'[1]	Switch to *ssnm*	Signon [2]	X'00C12217'[1]
SWITCH TO	IDENTIFY	Switch to *ssnm*	Signon [2]	CREATE THREAD
SIGNON, AUTH SIGNON, or CONTEXT SIGNON	X'00F30049'[1]	Switch to *ssnm*	Signon [2]	CREATE THREAD
CREATE THREAD	X'00F30049'[1]	Switch to *ssnm*	Signon [2]	X'00C12202'[1]
TERMINATE THREAD	X'00C12201'[1]	Switch to *ssnm*	Signon [2]	CREATE THREAD
IFI	X'00F30049'[1]	Switch to *ssnm*	Signon [2]	X'00C12202'[1]
SQL	X'00F30049'[1]	Switch to *ssnm*	X'00F30092'[1,3]	X'00C12202'[1]
SRRCMIT or SRRBACK	X'00F30049'[1]	Switch to *ssnm*	Signon [2]	X'00C12202'[1]

Notes:

1. Errors are identified by the Db2 reason code that RRSAF returns.
2. Signon means either the SIGNON function, the AUTH SIGNON function, or the CONTEXT SIGNON function.
3. The SIGNON, AUTH SIGNON, or CONTEXT SIGNON functions are not allowed if any SQL operations are requested after the CREATE THREAD function or after the last SRRCMIT or SRRBACK request.

The following table summarizes RRSAF behavior when the next call is an SQL statement or an IFI call or to the TERMINATE THREAD function, the TERMINATE IDENTIFY function, or the TRANSLATE function.

Table 20. Effect of call order when next call is SQL or IFI, TERMINATE THREAD, TERMINATE IDENTIFY, or TRANSLATE

	Next function			
Previous function	**SQL or IFI**	**TERMINATE THREAD**	**TERMINATE IDENTIFY**	**TRANSLATE**
Empty: first call	SQL or IFI call[4]	X'00C12204'[1]	X'00C12204'[1]	X'00C12204'[1]
IDENTIFY	SQL or IFI call[4]	X'00C12203'[1]	TERMINATE IDENTIFY	TRANSLATE
SWITCH TO	SQL or IFI call[4]	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE

Table 20. Effect of call order when next call is SQL or IFI, TERMINATE THREAD, TERMINATE IDENTIFY, or TRANSLATE (continued)

	Next function			
Previous function	**SQL or IFI**	**TERMINATE THREAD**	**TERMINATE IDENTIFY**	**TRANSLATE**
SIGNON, AUTH SIGNON, or CONTEXT SIGNON	SQL or IFI call[4]	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
CREATE THREAD	SQL or IFI call[4]	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
TERMINATE THREAD	SQL or IFI call [4]	X'00C12203'[1]	TERMINATE IDENTIFY	TRANSLATE
IFI	SQL or IFI call[4]	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
SQL	SQL or IFI call[4]	X'00F30093'[1,2]	X'00F30093'[1,3]	TRANSLATE
SRRCMIT or SRRBACK	SQL or IFI call[4]	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE

Notes:

1. Errors are identified by the Db2 reason code that RRSAF returns.
2. TERMINATE THREAD is not allowed if any SQL operations are requested after the CREATE THREAD function or after the last SRRCMIT or SRRBACK request.
3. TERMINATE IDENTIFY is not allowed if any SQL operations are requested after the CREATE THREAD function or after the last SRRCMIT or SRRBACK request.
4. If you are using an implicit connection to RRSAF and issue SQL or IFI calls, RRSAF issues implicit IDENTIFY and CREATE THREAD requests. If you continue with explicit RRSAF statements, you must follow the standard order of explicit RRSAF calls. Implicitly connecting to RRSAF does not cause an implicit SIGNON request. Therefore, you might need to issue an explicit SIGNON request to satisfy the standard order requirement. For example, an SQL statement followed by an explicit TERMINATE THREAD request results in an error. You must issue an explicit SIGNON request before issuing the TERMINATE THREAD request.

Related concepts

X'C1......' codes (Db2 Codes)
X'F3......' codes (Db2 Codes)

RRSAF connection functions

An Resource Recovery Services attachment facility (RRSAF) connection function specifies the action that you want RRSAF to take. You specify these functions when you invoke RRSAF through CALL DSNRLI statements.

Related concepts

CALL DSNRLI statement parameter list
The CALL DSNRLI statement explicitly invokes RRSAF. When you include CALL DSNRLI statements in your program, you must specify all parameters that precede the return code parameter.

Summary of RRSAF behavior
The effect of any Resource Recovery Services attachment facility (RRSAF) function depends in part on what functions the program has already run. You should plan the RRSAF function calls that your program makes to avoid any errors and major structural problems in your application.

IDENTIFY function for RRSAF

The RRSAF IDENTIFY function initializes a connection to Db2.

The IDENTIFY function establishes the caller's task as a user of Db2 services. If no other task in the address space currently is connected to the specified subsystem, the IDENTIFY function also initializes the address space to communicate with the Db2 address spaces. The IDENTIFY function establishes the cross-memory authorization of the address space to Db2 and builds address space control blocks.

The following diagram shows the syntax for the IDENTIFY function.

DSNRLI IDENTIFY function

```
>>- CALL DSNRLI -- ( -- function -- , -- ssnm -- , -- ribptr -- , -- eibptr -- , -- termecb -- , ->

>-- startecb ->

>--+---------------------------------------------------------------------------------+->
   '- , -- retcode --+-------------------------------------------------------------+-'
                     '- , -- reascode --+----------------------------------------+-'
                                        '- , -- groupoverride --+--------------+-'
                                                                '- , -- decpptr -'

>-- ) -><
```

Parameters point to the following areas:

function
An 18-byte area that contains IDENTIFY followed by 10 blanks.

ssnm
A 4-byte Db2 subsystem name, or group attachment or subgroup attachment name (if used in a data sharing group) to which the connection is made. If *ssnm* is less than four characters long, pad it on the right with blanks to a length of four characters.

ribptr
A 4-byte area in which RRSAF places the address of the release information block (RIB) after the call. You can use the RIB to determine the release level of the Db2 subsystem to which the application is connected. You can determine the modification level within the release level by examining the RIBCNUMB and RIBCINFO fields. If the value in the RIBCNUMB field is greater than zero, check the RIBCINFO field for modification levels.

If the RIB is not available (for example, if *ssnm* names a subsystem that does not exist), Db2 sets the 4-byte area to zeros.

The area to which *ribptr* points is below the 16-MB line.

This parameter is required. However, the application does not need to refer to the returned information.

eibptr
A 4-byte area in which RRSAF places the address of the environment information block (EIB) after the call. The EIB contains environment information, such as the data sharing group, the name of the Db2 member to which the IDENTIFY request was issued, and whether the subsystem is in new-function mode. If the Db2 subsystem is not in a data sharing group, RRSAF sets the data sharing group and member names to blanks. If the EIB is not available (for example, if *ssnm* names a subsystem that does not exist), RRSAF sets the 4-byte area to zeros.

The area to which *eibptr* points is above the 16-MB line.

This parameter is required. However, the application does not need to refer to the returned information.

termecb
The address of the application's event control block (ECB) that is used for Db2 termination. Db2 posts this ECB when the system operator enters the STOP DB2 command or when Db2 is terminating abnormally. Specify a value of 0 if you do not want to use a termination ECB.

The ECB is ignored when Db2 is already stopped. The application program must examine any nonzero RRSAF or Db2 reason codes before issuing a WAIT request on this ECB.

RRSAF puts a POST code in the ECB to indicate the type of termination as shown in the following table.

Table 21. Post codes for types of Db2 termination

POST code	Termination type
8	QUIESCE
12	FORCE
16	ABTERM

startecb

The address of the application's startup ECB. If Db2 has not started when the application issues the IDENTIFY call, Db2 posts the ECB when Db2 has started. If Db2 is already started, the startup ECB is ignored. and is not applied to the next Db2 startup. If Db2 is not started, and the startup ECB is queued, the termination ECB is ignored.

Enter a value of zero if you do not want to use a startup ECB. Db2 posts no more than one startup ECB per address space. The ECB that is posted is associated with the most recent IDENTIFY call from that address space. The application program must examine any nonzero RRSAF or Db2 reason codes before issuing a WAIT request on this ECB.

retcode

A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode

A 4-byte area in which RRSAF places a reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode* or its default. You can specify a default for *retcode* by specifying a comma or zero, depending on the language.

groupoverride

An 8-byte area that the application provides. This parameter is optional. If you do not want group attach to be attempted, specify 'NOGROUP'. This string indicates that the subsystem name that is specified by *ssnm* is to be used as a Db2 subsystem name, even if *ssnm* matches a group attachment or subgroup attachment name. If *groupoverride* is not provided, *ssnm* is used as the group attachment or subgroup attachment name if it matches a group attachment or subgroup attachment name.

If you specify this parameter in any language except assembler, you must also specify the *retcode* and *reascode* parameters. In assembler language, you can omit the *retcode* and *reascode* parameters by specifying commas as place-holders.

Recommendation: Avoid using the *groupoverride* parameter when possible, because it limits the ability to do dynamic workload routing in a Parallel Sysplex. However, you should use this parameter in a data sharing environment when you want to connect to a specific member of a data sharing group, and the subsystem name of that member is the same as the group attachment or subgroup attachment name.

decpptr

A 4-byte area in which RRSAF is to put the address of the DSNHDECP or a user-specified application defaults module that was loaded by subsystem *ssnm* when that subsystem was started. This 4-byte area is a 31-bit pointer. If *ssnm* is not found, the 4-byte area is set to 0.

The area to which *decpptr* points is above the 16-MB line.

If you specify this parameter in any language except assembler, you must also specify the *retcode*, *reascode*, and *groupoverride* parameters. In assembler language, you can omit the *retcode*, *reascode*, and *groupoverride* parameters by specifying commas as placeholders.

Example of RRSAF IDENTIFY function calls

The following table shows an IDENTIFY call in each language.

Table 22. Examples of RRSAF IDENTIFY calls

Language	Call example
Assembler	`CALL  DSNRLI,(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR)`
C[1]	`fnret=dsnrli(&idfyfn[0],&ssnm[0], &ribptr, &eibptr, &termecb, &startecb, &retcode, &reascode,&grpover[0],&decpptr);`
COBOL	`CALL  'DSNRLI' USING IDFYFN SSNM RIBTPR EIBPTR TERMECB STARTECB RETCODE REASCODE GRPOVER DECPPTR.`
Fortran	`CALL  DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR)`
PL/I[1]	`CALL  DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Internal processing for the IDENTIFY function

When you call the IDENTIFY function, Db2 performs the following steps:

1. Db2 determines whether the user address space is authorized to connect to Db2. Db2 invokes the z/OS SAF and passes a primary authorization ID to SAF. That authorization ID is the 7-byte user ID that is associated with the address space, unless an authorized function has built an ACEE for the address space. If an authorized function has built an ACEE, Db2 passes the 8-byte user ID from the ACEE.
   SAF calls an external security product, such as RACF, to determine if the task is authorized to use the following items:
   - The Db2 resource class (CLASS=DSNR)
   - The Db2 subsystem (SUBSYS=*ssnm*)
   - Connection type RRSAF
2. If that check is successful, Db2 calls the Db2 connection exit routine to perform additional verification and possibly change the authorization ID.
3. Db2 searches for a matching trusted context in the system cache and then the catalog based on the following criteria:
   - The primary authorization ID matches a trusted context SYSTEM AUTHID.
   - The job or started task name matches the JOBNAME attribute that is defined for the identified trusted context.

   If a trusted context is defined, Db2 checks if SECURITY LABEL is defined in the trusted context. If SECURITY LABEL is defined, Db2 verifies the SECURITY LABEL with RACF by using the RACROUTE VERIFY request. This security label is used to verify multi-level security for SYSTEM AUTHID.

   If a matching trusted context is defined, Db2 establishes the connection as trusted. Otherwise, the connection is established without any additional privileges.

4. Db2 then sets the connection name to RRSAF and the connection type to RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

SWITCH TO function for RRSAF

The RRSAF SWITCH TO function directs RRSAF, SQL, or IFI requests to a specified Db2 subsystem. Use the SWITCH TO function to establish connections to multiple Db2 subsystems from a single task.

The SWITCH TO function is useful only after a successful IDENTIFY call. If you have established a connection with one Db2 subsystem, you must issue a SWITCH TO call before you make an IDENTIFY call to another Db2 subsystem. Otherwise, Db2 returns return code X'200' and reason code X'00C12201'.

The first time that you make a SWITCH TO call to a new Db2 subsystem, Db2 returns return code 4 and reason code X'00C12205' as a warning to indicate that the current task has not yet been identified to the new Db2 subsystem.

The following diagram shows the syntax for the SWITCH TO function.

DSNRLI SWITCH TO function

```
>>- CALL DSNRLI -- ( -- function,ssnm -->

>--+------------------------------------------------------------+-- ) -><
   '- , -- retcode --+------------------------------------------+-'
                     '- , -- reascode --+-------------------------+-'
                                        '- , -- groupoverride --'
```

Parameters point to the following areas:

function
An 18-byte area that contains SWITCH TO followed by nine blanks.

ssnm
A 4-byte Db2 subsystem name, or group attachment or subgroup attachment name (if used in a data sharing group) to which the connection is made. If *ssnm* is less than four characters long, pad it on the right with blanks to a length of four characters.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify this parameter, you must also specify *retcode*.

groupoverride
An 8-byte area that the application provides. This parameter is optional. If you do not want group attach to be attempted, specify 'NOGROUP'. This string indicates that the subsystem name that is specified by *ssnm* is to be used as a Db2 subsystem name, even if *ssnm* matches a group attachment or subgroup attachment name. If *groupoverride* is not provided, *ssnm* is used as the group attachment or subgroup attachment name if it matches a group attachment or subgroup attachment name.

If you specify this parameter in any language except assembler, you must also specify the *retcode* and *reascode* parameters. In assembler language, you can omit the *retcode* and *reascode* parameters by specifying commas as place-holders.

Recommendation: Avoid using the *groupoverride* parameter when possible, because it limits the ability to do dynamic workload routing in a Parallel Sysplex. However, you should use this parameter in a data sharing environment when you want to connect to a specific member of a data sharing group, and the subsystem name of that member is the same as the group attachment or subgroup attachment name.

Examples of RRSAF SWITCH TO calls

The following table shows a SWITCH TO call in each language.

Table 23. Examples of RRSAF SWITCH TO calls

Language	Call example
Assembler	`CALL  DSNRLI,(SWITCHFN,SSNM,RETCODE,REASCODE,GRPOVER)`
C[1]	`fnret=dsnrli(&switchfn[0], &ssnm[0], &retcode, &reascode,&grpover[0]);`
COBOL	`CALL  'DSNRLI' USING SWITCHFN RETCODE REASCODE GRPOVER.`
Fortran	`CALL  DSNRLI(SWITCHFN,RETCODE,REASCODE,GRPOVER)`
PL/I[1]	`CALL  DSNRLI(SWITCHFN,RETCODE,REASCODE,GRPOVER);`

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Example of using the SWITCH TO function to interact with multiple Db2 subsystems

The following example shows how you can use the SWITCH TO function to interact with three Db2 subsystems.

```
RRSAF calls for subsystem db21:
  IDENTIFY
  SIGNON
  CREATE THREAD
Execute SQL on subsystem db21
SWITCH TO db22
IF retcode = 4 AND reascode = '00C12205'X THEN
  DO;
     RRSAF calls on subsystem db22:
     IDENTIFY
     SIGNON
     CREATE THREAD
  END;
Execute SQL on subsystem db22
SWITCH TO db23
IF retcode = 4 AND reascode = '00C12205'X THEN
  DO;
     RRSAF calls on subsystem db23:
     IDENTIFY
     SIGNON
     CREATE THREAD
  END;
Execute SQL on subsystem 23
SWITCH TO db21
Execute SQL on subsystem 21
SWITCH TO db22
```

```
Execute SQL on subsystem 22
SWITCH TO db21
Execute SQL on subsystem 21
SRRCMIT (to commit the UR)
SWITCH TO db23
Execute SQL on subsystem 23
SWITCH TO db22
Execute SQL on subsystem 22
SWITCH TO db21
Execute SQL on subsystem 21
SRRCMIT (to commit the UR)
```

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

SIGNON function for RRSAF

The RRSAF SIGNON function establishes a primary authorization ID and, optionally, one or more secondary authorization IDs for a connection.

Requirement: Your program does not need to be an authorized program to issue the SIGNON call. For that reason, before you issue the SIGNON call, you must issue the RACF external security interface macro RACROUTE REQUEST=VERIFY to perform the following actions:

- Define and populate an ACEE to identify the user of the program.
- Associate the ACEE with the user's TCB.
- Verify that the user is defined to RACF and authorized to use the application.

Generally, you issue a SIGNON call after an IDENTIFY call and before a CREATE THREAD call. You can also issue a SIGNON call if the application is at a point of consistency, and one of the following conditions is true:

- The value of *reuse* in the CREATE THREAD call was RESET.
- The value of *reuse* in the CREATE THREAD call was INITIAL, no held cursors are open, the package or plan is bound with KEEPDYNAMIC(NO), and all special registers are at their initial state. If open held cursors exist or the package or plan is bound with KEEPDYNAMIC(YES), you can issue a SIGNON call only if the primary authorization ID has not changed.

After you issue a SIGNON call, subsequent SQL statements return an error (SQLCODE -900) if the both of following conditions are true:

- The connection was established as trusted when it was initialized.
- The primary authorization ID that was used when you issued the SIGNON call is not allowed to use the trusted connection.

If a trusted context is defined, Db2 checks if SECURITY LABEL is defined in the trusted context. If SECURITY LABEL is defined, Db2 verifies the security label with RACF by using the RACROUTE VERIFY request. This security label is used to verify multi-level security for SYSTEM AUTHID.

The following diagram shows the syntax for the SIGNON function.

DSNRLI SIGNON function

```
>>- CALL DSNRLI -- ( -- function, correlation-id, -- accounting-token, accounting-interval -->

>--+----------------------------------------------------------------------------------+-><
   '-,retcode--+---------------------------------------------------------------------+-'
               '-,reascode--+-------------------------------------------------------+-'
                            '-,user--+----------------------------------------------+-'
                                     '-,appl--+-------------------------------------+-'
                                              '-,ws--+------------------------------+-'
                                                     '-,xid--+----------------------+-'
                                                             '-,accounting-string---'

>>- , --+-- user-length, user-longname --+-- , --+-- appl-length, appl-longname --+-><
        '------------ 0,0 ---------------'       '------------ 0,0 ---------------'

>>- , --+-- ws-length, ws-longname --+-- , --+-- correlation-length, correlation-longname --+-- ) -><
        '---------- 0,0 -------------'       '------------------ 0,0 -----------------------'
```

Parameters point to the following areas:

function
An 18-byte area that contains SIGNON followed by twelve blanks.

correlation-id
A 12-byte area in which you can put a Db2 correlation ID. The correlation ID is displayed in Db2 accounting and statistics trace records. You can use the correlation ID to correlate work units. This token appears in the output from the DISPLAY THREAD command. If you do not want to specify a correlation ID, fill the 12-byte area with blanks.

accounting-token
A 22-byte area in which you can put a value for a Db2 accounting token. This value is displayed in Db2 accounting and statistics trace records in the QWHCTOKN field, which is mapped by DSNDQWHC DSECT. Setting the value of the accounting token sets the value of the CURRENT CLIENT_ACCTNG special register. If *accounting-token* is less than 22 characters long, you must pad it on the right with blanks to a length of 22 characters. If you do not want to specify an accounting token, fill the 22-byte area with blanks.

Alternatively, you change the value of the Db2 accounting token with RRSAF functions AUTH SIGNON, CONTEXT SIGNON or SET_CLIENT_ID. You can retrieve the Db2 accounting token with the CURRENT CLIENT_ACCTNG special register only if the DDF accounting string is not set.

accounting-interval
A 6-byte area that specifies when Db2 writes an accounting record.

If you specify COMMIT in that area, Db2 writes an accounting record each time that the application issues SRRCMIT. This accounting record is written at the end of the second phase of a two-phase commit. If the accounting interval is COMMIT, and an SRRCMIT is issued while a held cursor is open, the accounting interval spans that commit and ends at the next valid accounting interval end point (such as the next SRRCMIT that is issued without open held cursors, application termination, or SIGNON with a new authorization ID).

If you specify any other value, Db2 writes an accounting record when the application terminates or when you call the SIGNON function with a new authorization ID.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode

A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify this parameter, you must also specify *retcode*.

user

A 16-byte area that contains the user ID of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the user ID sets the value of the CURRENT CLIENT_USERID special register. If *user* is less than 16 characters long, you must pad it on the right with blanks to a length of 16 characters.

This parameter is optional. If you specify *user*, you must also specify *retcode* and *reascode*. If you do not specify *user*, no user ID is associated with the connection.

appl

A 32-byte area that contains the application or transaction name of the user's application. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the application name sets the value of the CURRENT CLIENT_APPLNAME special register. If *appl* is less than 32 characters long, you must pad it on the right with blanks to a length of 32 characters.

This parameter is optional. If you specify *appl*, you must also specify *retcode*, *reascode*, and *user*. If you do not specify *appl*, no application or transaction is associated with the connection.

ws

An 18-byte area that contains the workstation name of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the workstation name sets the value of the CURRENT CLIENT_WRKSTNNAME special register. If *ws* is less than 18 characters long, you must pad it on the right with blanks to a length of 18 characters.

This field is optional. If you specify *ws*, you must also specify *retcode*, *reascode*, *user*, and *appl*. If you do not specify *ws*, no workstation name is associated with the connection.

xid

A 4-byte area that indicates whether the thread is part of a global transaction. A Db2 thread that is part of a global transaction can share locks with other Db2 threads that are part of the same global transaction and can access and modify the same data. A global transaction exists until one of the threads that is part of the global transaction is committed or rolled back.

You can specify one of the following values for *xid*:

0

Indicates that the thread is not part of a global transaction. The value 0 must be specified as a binary integer.

1

Indicates that the thread is part of a global transaction and that Db2 should retrieve the global transaction ID from RRS. If a global transaction ID already exists for the task, the thread becomes part of the associated global transaction. Otherwise, RRS generates a new global transaction ID. The value 1 must be specified as a binary integer. Alternatively, if you want Db2 to return the generated global transaction ID to the caller, specify an address instead of 1.

address

The 4-byte address of an area in which you enter a global transaction ID for the thread. If the global transaction ID already exists, the thread becomes part of the associated global transaction. Otherwise, RRS creates a new global transaction with the ID that you specify.

Alternatively, if you want Db2 to generate and return a global transaction ID, pass the address of a null global transaction ID by setting the *format ID* field of the global transaction ID to binary -1

('FFFFFFF'X). Db2 then replaces the contents of the area with the generated transaction ID. The area at the specified address must be in writable storage and have a length of at least 140 bytes to accommodate the largest possible transaction ID value.

The following table shows the format of a global transaction ID.

Table 24. Format of a user-created global transaction ID

Field description	Length in bytes	Data type
Format ID	4	Integer
Global transaction ID length (1 - 64)	4	Integer
Branch qualifier length (1 - 64)	4	Integer
Global transaction ID	1 to 64	Character
Branch qualifier	0 to 64	Character

accounting-string

A one-byte length field and a 255-byte area in which you can put a value for a Db2 accounting string. This value is placed in the DDF accounting trace records in the QMDASQLI field, which is mapped by DSNDQMDA DSECT. If *accounting-string* is less than 255 characters, you must pad it on the right with zeros to a length of 255 bytes. The entire 256 bytes is mapped by DSNDQMDA DSECT.

This parameter is optional. If you specify *accounting-string*, you must also specify *retcode*, *reascode*, *user*, *appl* and *xid*. If you do not specify *accounting-string*, no accounting string is associated with the connection.

You can also change the value of the accounting string with RRSAF functions AUTH SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID.

You can retrieve the DDF suffix portion of the accounting string with the CURRENT CLIENT_ACCTNG special register. The suffix portion of *accounting-string* can contain a maximum of 200 characters. The QMDASFLN field contains the accounting suffix length, and the QMDASUFX field contains the accounting suffix value. If the DDF accounting string is set, you cannot query the accounting token with the CURRENT CLIENT_ACCTNG special register.

The following parameters are optional and positional. These parameters override values specified earlier in the parameter list. To provide a value for a length, value pair, you must provide a value or specify a 0 length for previous parameters in the parameter list.

user-length, user-longname

A pair of parameters that consist of a 2-byte integer length and 128-byte string area. A comma separates the parameters. You can provide the user ID of the client user for accounting and monitoring purposes in *user-longname*. Db2 displays this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the user ID sets the value of the CURRENT CLIENT_USERID special register. Trailing blanks in *user-longname* are truncated and the length in *user-length* is updated.

These parameters are optional, to specify them you must also specify a value for *accounting-string*. A value of 0 in *user-length* skips processing of *user-longname*.

Important: These parameters override any value that is provided in *user*.

appl-length, appl-longname

A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters.You can provide the application or transaction name of the client user for accounting and monitoring purposes in *appl-longname*. Db2 displays this application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the application name sets the value of the CURRENT CLIENT_APPLNAME special register. Trailing blanks in *appl-longname* are truncated and the length in *appl-length* is updated.

These parameters are optional, to specify them you must also specify a value for *user-length, user-longname*. A value of 0 in *appl-length* skips processing of *appl-longname*.

Important: These parameters override any value that is provided in *appl*.

ws-length, ws-longname

A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters. You can provide the workstation name of the client user for accounting and monitoring purposes in *ws-longname*. Db2 displays this workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the workstation name sets the value of the CURRENT CLIENT_WRKSTNNAME special register. Trailing blanks in *ws-longname* are truncated and the length in *ws-length* is updated.

These parameters are optional, to specify them you must also specify a value for *appl-length, appl-longname*. A value of 0 in *ws-length* skips processing of *ws-longname*.

Important: These parameters override any value that is provided in *ws*.

correlation-length, correlation-longname

A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters. You can provide a unique value to correlate your business process names with Db2 threads in *correlation-longname*. Db2 displays this correlation token in the output from the DISPLAY THREAD DETAIL command. The CURRENT CLIENT_CORR_TOKEN special register contains the client correlation token. Trailing blanks in *correlation-longname* are truncated and the length in *correlation-length* is updated.

These parameters are optional, to specify them you must also specify a value for *ws-length, ws-longname*. A value of 0 in *correlation-length* skips processing of *correlation-longname*.

You can also change the value of the client correlation token with the RRSAF AUTH SIGNON function and the SET_CLIENT_ID function.

Example of RRSAF SIGNON calls

The following table shows a SIGNON call in each language.

Table 25. Examples of RRSAF SIGNON calls

Language	Call example
assembler	`CALL  DSNRLI,(SGNONFN,CORRID,ACCTTKN,ACCTINT,` `RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)`
C[1]	`fnret=dsnrli(&sgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &retcode, &reascode,` `&userid[0], &applname[0], &wsname[0], &xidptr);`

COBOL	`CALL  'DSNRLI' USING SGNONFN CORRID ACCTTKN ACCTINT RETCODE REASCODE USERID APPLNAME`
	`WSNAME`
	`XIDPTR.`

Fortran	`CALL  DSNRLI(SGNONFN,CORRID,ACCTTKN,ACCTINT,` `RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)`
PL/I[1]	`CALL  DSNRLI(SGNONFN,CORRID,ACCTTKN,ACCTINT,` `RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

The following example shows a SIGNON call in C [1] with all parameters passed in. Parameters that are numbers are passed in as integers and strings as character arrays. In this example, if *&useridlen* is larger than 0, then the value of CURRENT CLIENT_USERID special register is the value that is stored in *&luserid[0]*.

```
fnret=dsnrli(&sgnonfn[0],&corrid[0],&accttkn[0],&acctint[0],&retcode,&reascode,
&userid[0],&applname[0],&wsname[0],&xidptr,&lacctngid[0],
&useridlen,&luserid[0],&applidlen,&lapplid[0],&wsidlen,&lwsid[0],
&corrtkidlen,&lcorrtkid[0]);
```

Note:

1. For C applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

Related reference

RACROUTE REQUEST=VERIFY (standard form) (Security Server RACROUTE Macro Reference)

AUTH SIGNON function for RRSAF

The RRSAF AUTH SIGNON function enables an APF authorization program to pass an ID to Db2.

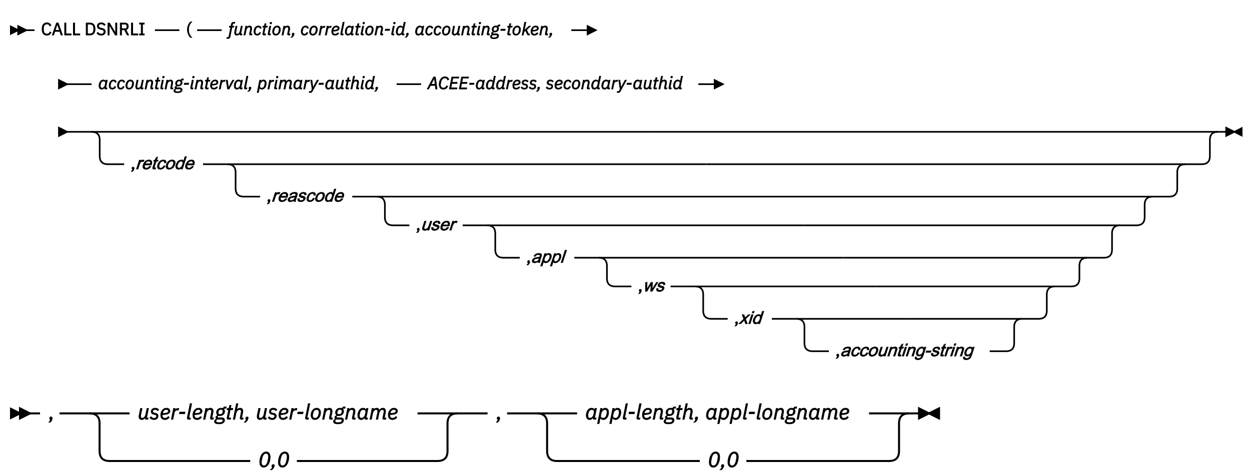
An APF-authorized program can pass to Db2 either a primary authorization ID and, optionally, one or more secondary authorization IDs, or an ACEE that is used for authorization checking. These IDs are then associated with the connection.

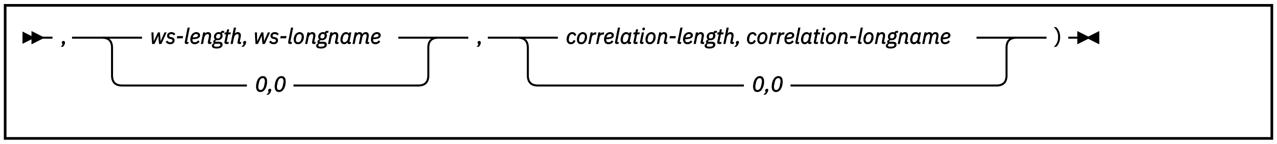
Generally, you issue an AUTH SIGNON call after an IDENTIFY call and before a CREATE THREAD call. You can also issue an AUTH SIGNON call if the application is at a point of consistency, and one of the following conditions is true:

- The value of *reuse* in the CREATE THREAD call was RESET.
- The value of *reuse* in the CREATE THREAD call was INITIAL, no held cursors are open, the package or plan is bound with KEEPDYNAMIC(NO), and all special registers are at their initial state. If open held cursors exist or the package or plan is bound with KEEPDYNAMIC(YES), a SIGNON call is permitted only if the primary authorization ID has not changed.

The following diagram shows the syntax for the AUTH SIGNON function.

DSNRLI AUTH SIGNON function





Parameters point to the following areas:

function
An 18-byte area that contains AUTH SIGNON followed by seven blanks.

correlation-id
A 12-byte area in which you can put a Db2 correlation ID. The correlation ID is displayed in Db2 accounting and statistics trace records. You can use the correlation ID to correlate work units. This token appears in output from the DISPLAY THREAD command. If you do not want to specify a correlation ID, fill the 12-byte area with blanks.

accounting-token
A 22-byte area in which you can put a value for a Db2 accounting token. This value is displayed in Db2 accounting and statistics trace records in the QWHCTOKN field, which is mapped by DSNDQWHC DSECT. Setting the value of the accounting token sets the value of the CURRENT CLIENT_ACCTNG special register. If *accounting-token* is less than 22 characters long, you must pad it on the right with blanks to a length of 22 characters. If you do not want to specify an accounting token, fill the 22-byte area with blanks.

You can also change the value of the Db2 accounting token with RRSAF functions SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID. You can retrieve the Db2 accounting token with the CURRENT CLIENT_ACCTNG special register only if the DDF accounting string is not set.

accounting-interval
A 6-byte area with that specifies when Db2 writes an accounting record.

If you specify COMMIT in that area, Db2 writes an accounting record each time that the application issues SRRCMIT. This accounting record is written at the end of the second phase of a two-phase commit. If the accounting interval is COMMIT, and an SRRCMIT is issued while a held cursor is open, the accounting interval spans that commit and ends at the next valid accounting interval end point (such as the next SRRCMIT that is issued without open held cursors, application termination, or SIGNON with a new authorization ID).

If you specify any other value, Db2 writes an accounting record when the application terminates or when you call the SIGNON function with a new authorization ID.

primary-authid
An 8-byte area in which you can put a primary authorization ID. If you are not passing the authorization ID to Db2 explicitly, put X'00' or a blank in the first byte of the area.

ACEE-address
The 4-byte address of an ACEE that you pass to Db2. If you do not want to provide an ACEE, specify 0 in this field.

secondary-authid
An 8-byte area in which you can put a secondary authorization ID. If you do not pass the authorization ID to Db2 explicitly, put X'00' or a blank in the first byte of the area. If you enter a secondary authorization ID, you must also enter a primary authorization ID.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascoder*, you must also specify *retcode*.

user

A 16-byte area that contains the user ID of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the user ID sets the value of the CURRENT CLIENT_USERID special register. If *user* is less than 16 characters long, you must pad it on the right with blanks to a length of 16 characters.

This parameter is optional. If you specify *user*, you must also specify *retcode* and *reascode*. If you do not specify this parameter, no user ID is associated with the connection.

appl

A 32-byte area that contains the application or transaction name of the user's application. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the application name sets the value of the CURRENT CLIENT_APPLNAME special register. If *appl* is less than 32 characters long, you must pad it on the right with blanks to a length of 32 characters.

This parameter is optional. If you specify *appl*, you must also specify *retcode*, *reascode*, and *user*. If you do not specify this parameter, no application or transaction is associated with the connection.

ws

An 18-byte area that contains the workstation name of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the workstation name sets the value of the CURRENT CLIENT_WRKSTNNAME special register. If *ws* is less than 18 characters long, you must pad it on the right with blanks to a length of 18 characters.

This parameter is optional. If you specify *ws*, you must also specify *retcode*, *reascode*, *user*, and *appl*. If you do not specify this parameter, no workstation name is associated with the connection.

You can also change the value of the workstation name with RRSAF functions SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID. You can retrieve the workstation name with the CURRENT CLIENT_WRKSTNNAME special register.

xid

A 4-byte area that indicates whether the thread is part of a global transaction. A Db2 thread that is part of a global transaction can share locks with other Db2 threads that are part of the same global transaction and can access and modify the same data. A global transaction exists until one of the threads that is part of the global transaction is committed or rolled back.

You can specify one of the following values for *xid*:

0

Indicates that the thread is not part of a global transaction. The value 0 must be specified as a binary integer.

1

Indicates that the thread is part of a global transaction and that Db2 should retrieve the global transaction ID from RRS. If a global transaction ID already exists for the task, the thread becomes part of the associated global transaction. Otherwise, RRS generates a new global transaction ID. The value 1 must be specified as a binary integer. Alternatively, if you want Db2 to return the generated global transaction ID to the caller, specify an address instead of 1.

address

The 4-byte address of an area into which you enter a global transaction ID for the thread. If the global transaction ID already exists, the thread becomes part of the associated global transaction. Otherwise, RRS creates a new global transaction with the ID that you specify.

Alternatively, if you want Db2 to generate and return a global transaction ID, pass the address of a null global transaction ID by setting the *format ID* field of the global transaction ID to binary -1 ('FFFFFFF'X). Db2 then replaces the contents of the area with the generated transaction ID. The area at the specified address must be in writable storage and have a length of at least 140 bytes to accommodate the largest possible transaction ID value.

The format of a global transaction ID is shown in the description of the RRSAF SIGNON function.

accounting-string
A 1-byte length field and a 255-byte area in which you can put a value for a Db2 accounting string. This value is placed in the DDF accounting trace records in the QMDASQLI field, which is mapped by DSNDQMDA DSECT. If *accounting-string* is less than 255 characters, you must pad it on the right with zeros to a length of 255 bytes. The entire 256 bytes is mapped by DSNDQMDA DSECT.

This parameter is optional. If you specify this *accounting-string*, you must also specify *retcode*, *reascode*, *user*, *appl*, and *xid*. If you do not specify this parameter, no accounting string is associated with the connection.

You can also change the value of the accounting string with RRSAF functions AUTH SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID.

You can retrieve the DDF suffix portion of the accounting string with the CURRENT CLIENT_ACCTNG special register. The suffix portion of *accounting-string* can contain a maximum of 200 characters. The QMDASFLN field contains the accounting suffix length, and the QMDASUFX field contains the accounting suffix value. If the DDF accounting string is set, you cannot query the accounting token with the CURRENT CLIENT_ACCTNG special register.

The following parameters are optional and positional. These parameters override values specified earlier in the parameter list. To provide a value for a length, value pair, you must provide a value or specify a 0 length for previous parameters in the parameter list.

user-length, user-longname
A pair of parameters that consist of a 2-byte integer length and 128-byte string area. A comma separates the parameters. You can provide the user ID of the client user for accounting and monitoring purposes in *user-longname*. Db2 displays this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the user ID sets the value of the CURRENT CLIENT_USERID special register. Trailing blanks in *user-longname* are truncated and the length in *user-length* is updated.

These parameters are optional, to specify them you must also specify a value for *accounting-string*. A value of 0 in *user-length* skips processing of *user-longname*.

Important: These parameters override any value that is provided in *user*.

appl-length, appl-longname
A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters. You can provide the application or transaction name of the client user for accounting and monitoring purposes in *appl-longname*. Db2 displays this application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the application name sets the value of the CURRENT CLIENT_APPLNAME special register. Trailing blanks in *appl-longname* are truncated and the length in *appl-length* is updated.

These parameters are optional, to specify them you must also specify a value for *user-length, user-longname*. A value of 0 in *appl-length* skips processing of *appl-longname*.

Important: These parameters override any value that is provided in *appl*.

ws-length, ws-longname
A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters. You can provide the workstation name of the client user for accounting and monitoring purposes in *ws-longname*. Db2 displays this workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the workstation name sets the value of the CURRENT CLIENT_WRKSTNNAME special register. Trailing blanks in *ws-longname* are truncated and the length in *ws-length* is updated.

These parameters are optional, to specify them you must also specify a value for *appl-length, appl-longname*. A value of 0 in *ws-length* skips processing of *ws-longname*.

Important: These parameters override any value that is provided in *ws*.

correlation-length, correlation-longname

A pair of parameters that consist of a 2-byte integer length and 255-byte string area. A comma separates the parameters. You can provide a unique value to correlate your business process names with Db2 threads in *correlation-longname*. Db2 displays this correlation token in the output from the DISPLAY THREAD DETAIL command. The CURRENT CLIENT_CORR_TOKEN special register contains the client correlation token. Trailing blanks in *correlation-longname* are truncated and the length in *correlation-length* is updated.

These parameters are optional, to specify them you must also specify a value for *ws-length, ws-longname*. A value of 0 in *correlation-length* skips processing of *correlation-longname*.

You can also change the value of the client correlation token with the RRSAF AUTH SIGNON function and the SET_CLIENT_ID function.

Example of RRSAF AUTH SIGNON calls

The following table shows a AUTH SIGNON call in each language.

Table 26. Examples of RRSAF AUTH SIGNON calls

Language	Call example
Assembler	`CALL  DSNRLI,(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPTR, SAUTHID,RETCODE,REASCODE, USERID,APPLNAME,WSNAME,XIDPTR)`
C[1]	`fnret=dsnrli(&asgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &pauthid[0], &aceeptr, &sauthid[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0], &xidptr);`
COBOL	`CALL  'DSNRLI' USING ASGNONFN CORRID ACCTTKN ACCTINT PAUTHID ACEEPTR SAUTHID RETCODE REASCODE USERID APPLNAME WSNAME XIDPTR.`
Fortran	`CALL  DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPTR, SAUTHID,RETCODE,REASCODE,USERID, APPLNAME,WSNAME,XIDPTR)`
PL/I[1]	`CALL  DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPTR, SAUTHID,RETCODE,REASCODE,USERID, APPLNAME,WSNAME,XIDPTR);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

The following example shows an AUTH SIGNON call in C [1] with all parameters passed in. Parameters that are numbers are passed in as integers and strings as character arrays. In this example, if *&useridlen* is larger than 0, then the value of CURRENT CLIENT_USERID special register is the value that is stored in *&luserid[0]*.

```
fnret = dsnrli(&authsgnfn[0],&corrid[0],&acctkn[0],&accint[0],&pauthid[0],
&aceeptr,&sauthid[0],&retcode,&reascode,&userid[0],&applname[0],
&wsname[0],&xidptr,&lacctngid[0],&useridlen,&luserid[0],&applidlen,
&lapplid[0],&wsidlen,&lwsid[0],&corrtkidlen,&lcorrtkid[0]);
```

Note:

1. For C applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

Related reference

SIGNON function for RRSAF
The RRSAF SIGNON function establishes a primary authorization ID and, optionally, one or more secondary authorization IDs for a connection.

CONTEXT SIGNON function for RRSAF

The RRSAF CONTEXT SIGNON function establishes a primary authorization ID and one or more secondary authorization IDs for a connection.

Requirement: Before you invoke CONTEXT SIGNON, you must have called the RRS context services function Set Context Data (CTXSDTA) to store a primary authorization ID and optionally, the address of an ACEE in the context data whose context key you supply as input to CONTEXT SIGNON.

The CONTEXT SIGNON function uses the context key to retrieve the primary authorization ID from data that is associated with the current RRS context. Db2 uses the RRS context services function Retrieve Context Data (CTXRDTA) to retrieve context data that contains the authorization ID and ACEE address. The context data must have the following format:

Version number
A 4-byte area that contains the version number of the context data. Set this area to 1.

Server product name
An 8-byte area that contains the name of the server product that set the context data.

ALET
A 4-byte area that can contain an ALET value. Db2 does not reference this area.

ACEE address
A 4-byte area that contains an ACEE address or 0 if an ACEE is not provided. Db2 requires that the ACEE is in the home address space of the task.

If you pass an ACEE address, the CONTEXT SIGNON function uses the value in ACEEGRPN as the secondary authorization ID if the length of the group name (ACEEGRPL) is not 0.

primary-authid
An 8-byte area that contains the primary authorization ID to be used. If the authorization ID is less than 8 bytes in length, pad it on the right with blank characters to a length of 8 bytes.

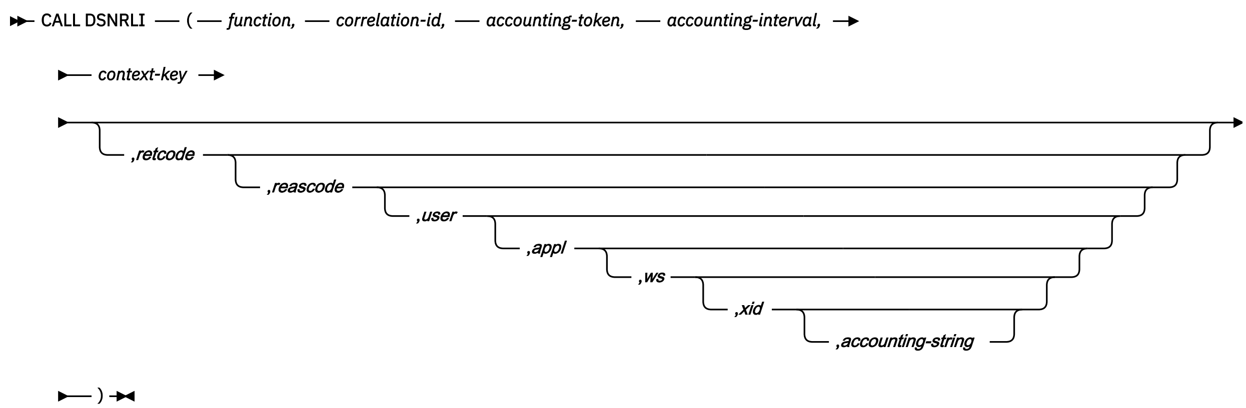
If the new primary authorization ID is not different than the current primary authorization ID (which was established when the IDENTIFY function was invoked or at a previous SIGNON invocation), Db2 invokes only the signon exit. If the value has changed, Db2 establishes a new primary authorization ID and new SQL authorization ID and then invokes the signon exit.

Generally, you issue a CONTEXT SIGNON call after an IDENTIFY call and before a CREATE THREAD call. You can also issue a CONTEXT SIGNON call if the application is at a point of consistency, and one of the following conditions is true:

- The value of *reuse* in the CREATE THREAD call was RESET.
- The value of *reuse* in the CREATE THREAD call was INITIAL, no held cursors are open, the package or plan is bound with KEEPDYNAMIC(NO), and all special registers are at their initial state. If open held cursors exist or the package or plan is bound with KEEPDYNAMIC(YES), a SIGNON call is permitted only if the primary authorization ID has not changed.

The following diagram shows the syntax for the CONTEXT SIGNON function.

DSNRLI CONTEXT SIGNON function



Parameters point to the following areas:

function
: An 18-byte area that contains CONTEXT SIGNON followed by four blanks.

correlation-id
: A 12-byte area in which you can put a Db2 correlation ID. The correlation ID is displayed in Db2 accounting and statistics trace records. You can use the correlation ID to correlate work units. This token appears in output from the DISPLAY THREAD command. If you do not want to specify a correlation ID, fill the 12-byte area with blanks.

accounting-token
: A 22-byte area in which you can put a value for a Db2 accounting token. This value is displayed in Db2 accounting and statistics trace records in the QWHCTOKN field, which is mapped by DSNDQWHC DSECT. Setting the value of the accounting token sets the value of the CURRENT CLIENT_ACCTNG special register. If *accounting-token* is less than 22 characters long, you must pad it on the right with blanks to a length of 22 characters. If you do not want to specify an accounting token, fill the 22-byte area with blanks.

  You can also change the value of the Db2 accounting token with RRSAF functions SIGNON, AUTH SIGNON, or SET_CLIENT_ID. You can retrieve the Db2 accounting token with the CURRENT CLIENT_ACCTNG special register only if the DDF accounting string is not set.

accounting-interval
: A 6-byte area that specifies when Db2 writes an accounting record.

  If you specify COMMIT in that area, Db2 writes an accounting record each time that the application issues SRRCMIT. This accounting record is written at the end of the second phase of a two-phase commit. If the accounting interval is COMMIT, and an SRRCMIT is issued while a held cursor is open, the accounting interval spans that commit and ends at the next valid accounting interval end point (such as the next SRRCMIT that is issued without open held cursors, application termination, or SIGNON with a new authorization ID).

  If you specify any other value, Db2 writes an accounting record when the application terminates or when you call the SIGNON function with a new authorization ID.

context-key
: A 32-byte area in which you put the context key that you specified when you called the RRS Set Context Data (CTXSDTA) service to save the primary authorization ID and an optional ACEE address.

retcode
: A 4-byte area in which RRSAF places the return code.

  This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode

A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

user

A 16-byte area that contains the user ID of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the user ID sets the value of the CURRENT CLIENT_USERID special register. If *user* is less than 16 characters long, you must pad it on the right with blanks to a length of 16 characters.

This parameter is optional. If you specify *user*, you must also specify *retcode* and *reascode*. If you do not specify *user*, no user ID is associated with the connection.

appl

A 32-byte area that contains the application or transaction name of the user's application. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the application name sets the value of the CURRENT CLIENT_APPLNAME special register. If *appl* is less than 32 characters long, you must pad it on the right with blanks to a length of 32 characters.

This parameter is optional. If you specify *appl*, you must also specify *retcode*, *reascode*, and *user*. If you do not specify *appl*, no application or transaction is associated with the connection.

ws

An 18-byte area that contains the workstation name of the client user. You can use this parameter to provide the identity of the client user for accounting and monitoring purposes. Db2 displays the workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. Setting the workstation name sets the value of the CURRENT CLIENT_WRKSTNNAME special register. If *ws* is less than 18 characters long, you must pad it on the right with blanks to a length of 18 characters.

This parameter is optional. If you specify *ws*, you must also specify *retcode*, *reascode*, *user*, and *appl*. If you do not specify *ws*, no workstation name is associated with the connection.

You can also change the value of the workstation name with the RRSAF functions SIGNON, AUTH SIGNON, or SET_CLIENT_ID. You can retrieve the workstation name with the CLIENT_WRKSTNNAME special register.

xid

A 4-byte area that indicates whether the thread is part of a global transaction. A Db2 thread that is part of a global transaction can share locks with other Db2 threads that are part of the same global transaction and can access and modify the same data. A global transaction exists until one of the threads that is part of the global transaction is committed or rolled back.

You can specify one of the following values for *xid*:

0

Indicates that the thread is not part of a global transaction. The value 0 must be specified as a binary integer.

1

Indicates that the thread is part of a global transaction and that Db2 should retrieve the global transaction ID from RRS. If a global transaction ID already exists for the task, the thread becomes part of the associated global transaction. Otherwise, RRS generates a new global transaction ID. The value 1 must be specified as a binary integer. Alternatively, if you want Db2 to return the generated global transaction ID to the caller, specify an address instead of 1.

address
The 4-byte address of an area into which you enter a global transaction ID for the thread. If the global transaction ID already exists, the thread becomes part of the associated global transaction. Otherwise, RRS creates a new global transaction with the ID that you specify.

Alternatively, if you want Db2 to generate and return a global transaction ID, pass the address of a null global transaction ID by setting the *format ID* field of the global transaction ID to binary -1 ('FFFFFFF'X). Db2 then replaces the contents of the area with the generated transaction ID. The area at the specified address must be in writable storage and have a length of at least 140 bytes to accommodate the largest possible transaction ID value.

The format of a global transaction ID is shown in the description of the RRSAF SIGNON function.

accounting-string
A one-byte length field and a 255-byte area in which you can put a value for a Db2 accounting string. This value is placed in the DDF accounting trace records in the QMDASQLI field, which is mapped by DSNDQMDA DSECT. If *accounting-string* is less than 255 characters, you must pad it on the right with zeros to a length of 255 bytes. The entire 256 bytes is mapped by DSNDQMDA DSECT.

This parameter is optional. If you specify this *accounting-string*, you must also specify *retcode*, *reascode*, *user*, *appl* and *xid*. If you do not specify this parameter, no accounting string is associated with the connection.

You can also change the value of the accounting string with RRSAF functions AUTH SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID.

You can retrieve the DDF suffix portion of the accounting string with the CURRENT CLIENT_ACCTNG special register. The suffix portion of *accounting-string* can contain a maximum of 200 characters. The QMDASFLN field contains the accounting suffix length, and the QMDASUFX field contains the accounting suffix value. If the DDF accounting string is set, you cannot query the accounting token with the CURRENT CLIENT_ACCTNG special register.

Example of RRSAF CONTEXT SIGNON calls

The following table shows a CONTEXT SIGNON call in each language.

Table 27. Examples of RRSAF CONTEXT SIGNON calls

Language	Call example
Assembler	`CALL  DSNRLI,(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXTKEY, RETCODE,REASCODE,USERID,APPLNAME, WSNAME,XIDPTR)`
C[1]	`fnret=dsnrli(&csgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &ctxtkey[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0], &xidptr);`
COBOL	`CALL  'DSNRLI' USING CSGNONFN CORRID ACCTTKN ACCTINT CTXTKEY RETCODE REASCODE USERID APPLNAME WSNAME XIDPTR.`
Fortran	`CALL  DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXTKEY, RETCODE,REASCODE, USERID,APPLNAME, WSNAME,XIDPTR)`
PL/I[1]	`CALL  DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXTKEY, RETCODE,REASCODE,USERID,APPLNAME, WSNAME,XIDPTR);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

Related reference

SIGNON function for RRSAF
The RRSAF SIGNON function establishes a primary authorization ID and, optionally, one or more secondary authorization IDs for a connection.

SET_ID function for RRSAF

The RRSAF SET_ID function sets a new value for the client program ID that can be used to identify the user. The function then passes this information to Db2 when the next SQL request is processed.

The following diagram shows the syntax of the SET_ID function.

DSNRLI SET_ID function

CALL DSNRLI (*function*, *program-id* [, *retcode* [, *reascode*]])

Parameters point to the following areas:

function
An 18-byte area that contains SET_ID followed by 12 blanks.

program-id
An 80-byte area that contains the caller-provided string to be passed to Db2. If *program-id* is less than 80 characters, you must pad it with blanks on the right to a length of 80 characters.

Db2 places the contents of *program-id* into IFCID 316 records, along with other statistics, so that you can identify which program is associated with a particular SQL statement.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode* RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

Example of RRSAF SET_ID calls

The following table shows a SET_ID call in each language.

Table 28. Examples of RRSAF SET_ID calls

Language	Call example
Assembler	`CALL  DSNRLI,(SETIDFN,PROGID,RETCODE,REASCODE)`
C[1]	`fnret=dsnrli(&setidfn[0], &progid[0], &retcode, &reascode);`

Table 28. Examples of RRSAF SET_ID calls (continued)

Language	Call example
COBOL	`CALL 'DSNRLI' USING SETIDFN PROGID RETCODE REASCODE.`
Fortran	`CALL DSNRLI(SETIDFN,PROGID,RETCODE,REASCODE)`
PL/I[1]	`CALL DSNRLI(SETIDFN,PROGID,RETCODE,REASCODE);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

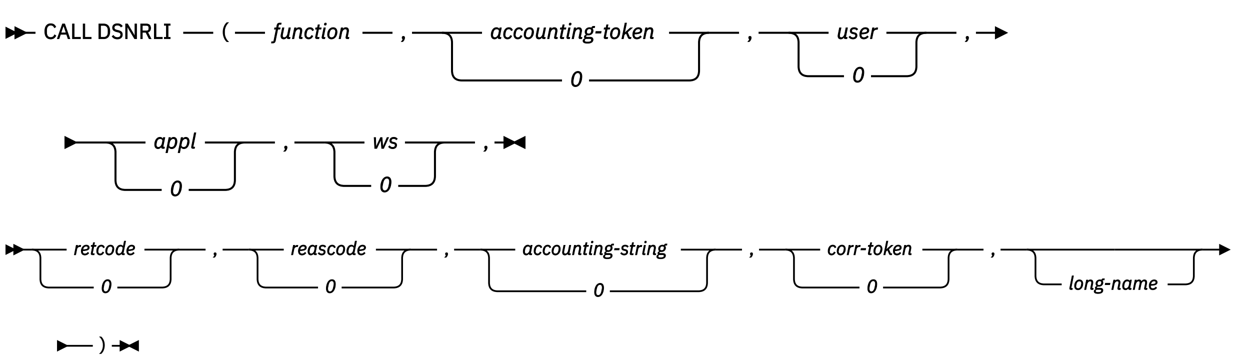
SET_CLIENT_ID function for RRSAF

The RRSAF SET_CLIENT_ID function sets new values for the client user ID, the application program name, the workstation name, the accounting token, the DDF client accounting string, the correlation token, and the long name. The function then passes this information to Db2 when the next SQL request is processed.

These values can be used to identify the end user. The calling program defines the contents of these parameters. Db2 places the parameter values in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records.

The following diagram shows the syntax of the SET_CLIENT_ID function.

DSNRLI SET_CLIENT_ID function



Parameters point to the following areas:

function
An 18-byte area that contains SET_CLIENT_ID followed by 5 blanks.

accounting-token
A 22-byte area in which you can put a value for a Db2 accounting token. This value is placed in the Db2 accounting and statistics trace records in the QWHCTOKN field, which is mapped by DSNDQWHC DSECT. If *accounting-token* is less than 22 characters long, you must pad it on the right with blanks to a length of 22 characters.

You can omit this parameter by specifying a value of 0 in the parameter list.

Alternatively, you can change the value of the Db2 accounting token with the RRSAF functions SIGNON, AUTH SIGNON, or CONTEXT SIGNON. You can retrieve the Db2 accounting token with the CURRENT CLIENT_ACCTNG special register only if the DDF accounting string is not set.

user

A 16-byte or 128-byte area that contains the user ID of the client end user. You can use this parameter to provide the identity of the client end user for accounting and monitoring purposes. Db2 places this user ID in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. If *user* is less than 16 characters long, you must pad it on the right with blanks to a length of 16 characters.

You can omit this parameter by specifying a value of 0 in the parameter list.

If the *long-name* parameter is specified, the maximum length of the *user* parameter is 128 bytes. If *user* is less than 128 characters long, you must pad it on the right with blanks to a length of 128 characters.

You can also change the value of the client user ID with the RRSAF functions SIGNON, AUTH SIGNON, or CONTEXT SIGNON. You can retrieve the client user ID with the CLIENT_USERID special register.

appl

An 32-byte or 255-byte area that contains the application or transaction name of the end user's application. You can use this parameter to provide the identity of the client end user for accounting and monitoring purposes. Db2 places the application name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. If *appl* is less than 32 characters, you must pad it on the right with blanks to a length of 32 characters.

You can omit this parameter by specifying a value of 0 in the parameter list.

If the *long-name* parameter is specified, the maximum length of the *appl* parameter is 255 bytes. If *appl* is less than 255 characters long, you must pad it on the right with blanks to a length of 255 characters.

You can also change the value of the application name with the RRSAF functions SIGNON, AUTH SIGNON, or CONTEXT SIGNON. You can retrieve the application name with the CLIENT_APPLNAME special register.

ws

An 18-byte or 255-byte area that contains the workstation name of the client end user. You can use this parameter to provide the identity of the client end user for accounting and monitoring purposes. Db2 places this workstation name in the output from the DISPLAY THREAD command and in Db2 accounting and statistics trace records. If *ws* is less than 18 characters, you must pad it on the right with blanks to a length of 18 characters.

You can omit this parameter by specifying a value of 0 in the parameter list.

If the *long-name* parameter is specified, the maximum length of the *ws* parameter is 255 bytes. If *ws* is less than 255 characters long, you must pad it on the right with blanks to a length of 255 characters.

You can also change the value of the workstation name with the RRSAF functions SIGNON, AUTH SIGNON, or CONTEXT SIGNON. You can retrieve the workstation name with the CLIENT_WRKSTNNAME special register.

retcode

A 4-byte area in which RRSAF places the return code.

You can omit this parameter by specifying a value of 0 in the parameter list.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode

A 4-byte area in which RRSAF places the reason code.

You can omit this parameter by specifying a value of 0 in the parameter list.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

accounting-string
A one-byte length field and a 255-byte area in which you can put a value for a Db2 accounting string. This value is placed in the DDF accounting trace records in the QMDASUFX field, which is mapped by DSNDQMDA DSECT. If *accounting-string* is less than 255 characters, you must pad it on the right with zeros to a length of 255 bytes. The entire 256 bytes is mapped by DSNDQMDA DSECT.

You can omit this parameter by specifying a value of 0 in the parameter list.

This parameter is optional. If you specify this *accounting-string*, you must also specify *retcode*, *reascode*, *user*, and *appl*. If you do not specify this parameter, no accounting string is associated with the connection.

You can also change the value of the accounting string with RRSAF functions AUTH SIGNON, CONTEXT SIGNON, or SET_CLIENT_ID.

You can retrieve the DDF suffix portion of the accounting string with the CURRENT CLIENT_ACCTNG special register. The suffix portion of *accounting-string* can contain a maximum of 200 characters. The QMDASFLN field contains the accounting suffix length, and the QMDASUFX field contains the accounting suffix value. If the DDF accounting string is set, you cannot query the accounting token with the CURRENT CLIENT_ACCTNG special register.

corr-token
An 255-byte area where you specify a client correlation token. You can specify a unique value to correlate your business process within Db2 and your entire business enterprise. The value of *corr-token* is displayed by the DISPLAY THREAD DETAIL command. The CURRENT CLIENT_CORR_TOKEN special register contains the client correlation token. If *corr-token* is less than 255 characters, you must pad it on the right with blanks to a length of 255 bytes.

You can omit this parameter by specifying a value of 0 in the parameter list. If you specify *corr-token* you must also specify *long-name*.

You can also change the value of the client correlation token with the RRSAF SIGNON function.

long-name
An 8-byte area that contains the value LONGNAME.

This optional parameter is used to indicate to the RRSAF function that the input parameters *user*, *appl*, *ws*, *accounting-string*, and *corr-token* can accept longer lengths. You cannot selectively associate the *long-name* parameter with any individual parameter.

Example of RRSAF SET_CLIENT_ID calls

The following table shows a SET_CLIENT_ID call in each language.

Table 29. Examples of RRSAF SET_CLIENT_ID calls

Language	Call example
Assembler	`CALL  DSNRLI,(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE,` `ACCOUNTINGSTRING,CORRTOKEN,LONGNAME)`
C[1]	`fnret=dsnrli(&seclidfn[0], &acct[0], &user[0], &appl[0], &ws[0], &retcode,` `&reascode, &accountingstring[0], &corrtoken[0], &longname[0]);`
COBOL	`CALL  'DSNRLI' USING SECLIDFN ACCT USER APPL WS RETCODE REASCODE` `ACCOUNTING-STRING CORR-TOKEN LONG-NAME.`

Table 29. Examples of RRSAF SET_CLIENT_ID calls (continued)

Language	Call example
Fortran	`CALL DSNRLI(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE, ACCOUNTINGSTRING,CORRTOKEN,LONGNAME)`
PL/I[1]	`CALL DSNRLI(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE, ACCOUNTINGSTRING,CORRTOKEN,LONGNAME);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility

The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

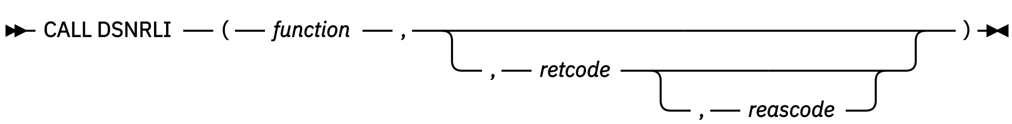
SET_REPLICATION function for RRSAF

The RRSAF SET_REPLICATION function enables an APF authorized program to identify to Db2 as a replication program.

Calling the SET_REPLICATION function is optional. If you do not call it, Db2 treats the application normally. The SET_REPLICATION function allows the application to perform insert, update, and delete operations then the tablespace or database is started access RREPL.

The following diagram shows the syntax for the SET REPLICATION function.

DSNRLI SET_REPLICATION function



Parameters point to the following areas:

function
An 18-byte area that contains SET_REPLICATION.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places a reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

Related tasks

Invoking the Resource Recovery Services attachment facility

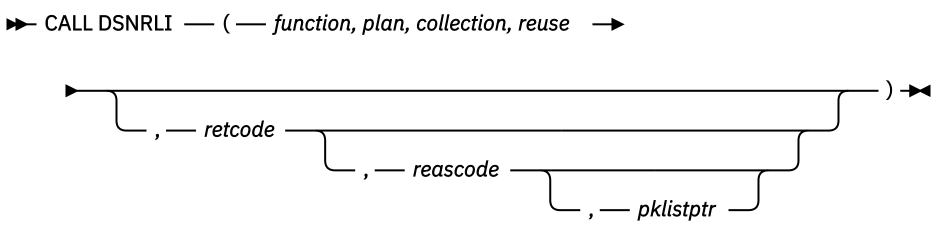
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

CREATE THREAD function for RRSAF

The RRSAF CREATE THREAD function allocates the Db2 resources that are required for an application to issue SQL or IFI requests. This function must complete before the application can execute SQL statements or IFI requests.

The following diagram shows the syntax of the CREATE THREAD function.

DSNRLI CREATE THREAD function



Parameters point to the following areas:

function
An 18-byte area that contains CREATE THREAD followed by five blanks.

plan
An 8-byte Db2 plan name. RRSAF allocates the named plan.

If you provide a collection name instead of a plan name, specify the question mark character (?) in the first byte of this field. Db2 then allocates a special plan named ?RRSAF and uses the value that you specify for *collection* . When Db2 allocates a plan named ?RRSAF, Db2 checks authorization to execute the package in the same way as it checks authorization to execute a package from a requester other than Db2 for z/OS.

If you do not provide a collection name in the *collection* field, you must enter a valid plan name in this field.

collection
An 18-byte area in which you enter a collection name. Db2 uses the collection names to locate a package that is associated with the first SQL statement in the program.

When you provide a collection name and put the question mark character (?) in the *plan* field, Db2 allocates a plan named ?RRSAF and a package list that contains the following two entries:

- The specified collection name.
- An entry that contains * for the location, collection name, and package name. (This entry lets the application access remote locations and access packages in collections other than the default collection that is specified at create thread time.)

The application can use the SET CURRENT PACKAGESET statement to change the collection ID that Db2 uses to locate a package.

If you provide a plan name in the *plan* field, Db2 ignores the value in the *collection* field.

reuse
An 8-byte area that controls the action that Db2 takes if a SIGNON call is issued after a CREATE THREAD call. Specify one of the following values in this field:

RESET
Releases any held cursors and reinitializes the special registers

INITIAL
Does not allow the SIGNON call

This parameter is required. If the 8-byte area does not contain either RESET or INITIAL, the default value is INITIAL.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

pklistptr
A 4-byte field that contains a pointer to a user-supplied data area that contains a list of collection IDs. A collection ID is an SQL identifier of 1 to 128 letters, digits, or the underscore character that identifies a collection of packages. The length of the data area is a maximum of 2050 bytes. The data area contains a 2-byte length field, followed by up to 2048 bytes of collection ID entries, separated by commas.

When you specify *pklistptr* and the question mark character (?) in the *plan* field, Db2 allocates a special plan named ?RRSAF and a package list that contains the following entries:

- The collection names that you specify in the data area to which pklistptr points
- An entry that contains * for the location, collection ID, and package name

If you also specify *collection*, Db2 ignores that value.

Each collection entry must be of the form *collection-ID*.*, *.*collection-ID*.*, or *.*.*. *collection-ID* and must follow the naming conventions for a collection ID, as described in the description of the BIND and REBIND options.

Db2 uses the collection names to locate a package that is associated with the first SQL statement in the program. The entry that contains *.*.* lets the application access remote locations and access packages in collections other than the default collection that is specified at create thread time.

The application can use the SET CURRENT PACKAGESET statement to change the collection ID that Db2 uses to locate a package.

This parameter is optional. If you specify this parameter, you must also specify *retcode* and *reascode*.

If you provide a plan name in the *plan* field, Db2 ignores the *pklistptr* value.

Recommendation: Using a package list can have a negative impact on performance. For better performance, specify a short package list.

Example of RRSAF CREATE THREAD calls

The following table shows a CREATE THREAD call in each language.

Table 30. Examples of RRSAF CREATE THREAD calls

Language	Call example
Assembler	`CALL DSNRLI,(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLISTPTR)`
C[1]	`fnret=dsnrli(&crthrdfn[0], &plan[0], &collid[0], &reuse[0], &retcode, &reascode, &pklistptr);`

Table 30. Examples of RRSAF CREATE THREAD calls (continued)

Language	Call example
COBOL	`CALL  'DSNRLI' USING CRTHRDFN PLAN COLLID REUSE RETCODE REASCODE PKLSTPTR.`
Fortran	`CALL  DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLSTPTR)`
PL/I[1]	`CALL  DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLSTPTR);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

Authorizing plan or package access through applications (Managing Security)

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

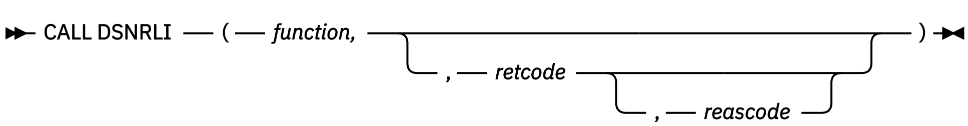
TERMINATE THREAD function for RRSAF

The RRSAF TERMINATE THREAD function deallocates Db2 resources that are associated with a plan and were previously allocated for an application by the CREATE THREAD function. You can then use the CREATE THREAD function to allocate another plan with the same connection.

If you call the TERMINATE THREAD function and the application is not at a point of consistency, RRSAF returns reason code X'00C12211'.

The following diagram shows the syntax of the TERMINATE THREAD function.

DSNRLI TERMINATE THREAD function



Parameters point to the following areas:

function
An 18-byte area the contains TERMINATE THREAD followed by two blanks.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

Example of RRSAF TERMINATE THREAD calls

The following table shows a TERMINATE THREAD call in each language.

Table 31. Examples of RRSAF TERMINATE THREAD calls

Language	Call example
Assembler	`CALL  DSNRLI,(TRMTHDFN,RETCODE,REASCODE)`
C[1]	`fnret=dsnrli(&trmthdfn[0], &retcode, &reascode);`
COBOL	`CALL  'DSNRLI' USING TRMTHDFN RETCODE REASCODE.`
Fortran	`CALL  DSNRLI(TRMTHDFN,RETCODE,REASCODE)`
PL/I[1]	`CALL  DSNRLI(TRMTHDFN,RETCODE,REASCODE);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

TERMINATE IDENTIFY function for RRSAF

The RRSAF TERMINATE IDENTIFY function terminates a connection to Db2. Calling the TERMINATE IDENTIFY function is optional. If you do not call it, Db2 performs the same functions when the task terminates.

If Db2 terminates, the application must issue TERMINATE IDENTIFY to reset the RRSAF control blocks. This action ensures that future connection requests from the task are successful when Db2 restarts.

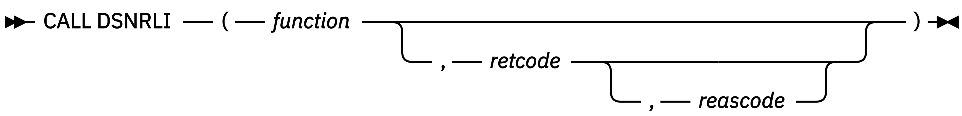
The TERMINATE IDENTIFY function removes the calling task's connection to Db2. If no other task in the address space has an active connection to Db2, Db2 also deletes the control block structures that were created for the address space and removes the cross-memory authorization.

If the application is not at a point of consistency when you call the TERMINATE IDENTIFY function, RRSAF returns reason code X'00C12211'.

If the application allocated a plan, and you call the TERMINATE IDENTIFY function without first calling the TERMINATE THREAD function, Db2 deallocates the plan before terminating the connection.

The following diagram shows the syntax of the TERMINATE IDENTIFY function.

DSNRLI TERMINATE IDENTIFY function



Parameters point to the following areas:

function
An 18-byte area that contains TERMINATE IDENTIFY.

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

Example of RRSAF TERMINATE IDENTIFY calls

The following table shows a TERMINATE IDENTIFY call in each language.

Table 32. Examples of RRSAF TERMINATE IDENTIFY calls

Language	Call example
Assembler	`CALL  DSNRLI,(TMIDFYFN,RETCODE,REASCODE)`
C[1]	`fnret=dsnrli(&tmidfyfn[0], &retcode, &reascode);`
COBOL	`CALL  'DSNRLI' USING TMIDFYFN RETCODE REASCODE.`
Fortran	`CALL  DSNRLI(TMIDFYFN,RETCODE,REASCODE)`
PL/I[1]	`CALL  DSNRLI(TMIDFYFN,RETCODE,REASCODE);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

Invoking the Resource Recovery Services attachment facility

The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

TRANSLATE function for RRSAF

The RRSAF TRANSLATE function converts a hexadecimal reason code for a Db2 error into a signed integer SQL code and a printable error message. The SQL code and message text are placed in the SQLCODE and SQLSTATE host variables or related fields of the SQLCA.

Consider the following rules and recommendations about when to use and not use the TRANSLATE function:

- You cannot call the TRANSLATE function from the Fortran language.
- Call the TRANSLATE function only after a successful IDENTIFY operation. For errors that occur during SQL or IFI requests, the TRANSLATE function performs automatically.
- The TRANSLATE function translates codes that begin with X'00F3', but it does not translate RRSAF reason codes that begin with X'00C1'.

If you receive error reason code X'00F30040' (resource unavailable) after an OPEN request, the TRANSLATE function returns the name of the unavailable database object in the last 44 characters of the SQLERRM field.

If the TRANSLATE function does not recognize the error reason code, it returns SQLCODE -924 (SQLSTATE '58006') and places a printable copy of the original Db2 function code and the return and error reason codes in the SQLERRM field. The contents of registers 0 and 15 do not change, unless TRANSLATE fails. In this case, register 0 is set to X'00C12204', and register 15 is set to 200.

The following diagram shows the syntax of the TRANSLATE function.

DSNRLI TRANSLATE function

```
>>- CALL DSNRLI -- ( -- function, sqlca --+---------------------------------------+-- ) -><
                                          '- , -- retcode --+------------------+-'
                                                            '- , -- reascode --'
```

Parameters point to the following areas:

function
An 18-byte area that contains the word TRANSLATE followed by nine blanks.

sqlca
The program's SQL communication area (SQLCA).

retcode
A 4-byte area in which RRSAF places the return code.

This parameter is optional. If you do not specify *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF places the reason code.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

If you specify *reascode*, you must also specify *retcode*.

Example of RRSAF TRANSLATE calls

The following table shows a TRANSLATE call in each language.

Table 33. Examples of RRSAF TRANSLATE calls

Language	Call example
Assembler	`CALL  DSNRLI,(XLATFN,SQLCA,RETCODE,REASCODE)`
C[1]	`fnret=dsnrli(&connfn[0], &sqlca, &retcode, &reascode);`
COBOL	`CALL  'DSNRLI' USING XLATFN SQLCA RETCODE REASCODE.`
PL/I[1]	`CALL  DSNRLI(XLATFN,SQLCA,RETCODE,REASCODE);`

Note:

1. For C, C++, and PL/I applications, you must include the appropriate compiler directives, because DSNRLI is an assembler language program. These compiler directives are described in the instructions for invoking RRSAF.

Related tasks

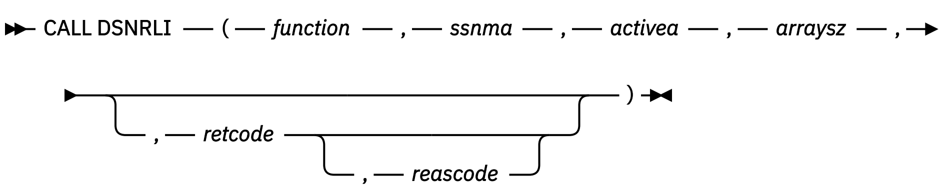
Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

FIND_DB2_SYSTEMS function for RRSAF

The RRSAF FIND_DB2_SYSTEMS function identifies all active Db2 subsystems on a z/OS LPAR.

The following diagram shows the syntax of the FIND_DB2_SYSTEMS function.

DSNRLI FIND_DB2_SYSTEMS function



Parameters point to the following areas:

function
An 18-byte area that contains FIND_DB2_SYSTEMS followed by two blanks.

ssnma
A storage area for an array of 4-byte character strings into which RRSAF places the names of all the Db2 subsystems (SSIDs) that are defined for the current LPAR. You must provide the storage area. If the array is larger than the number of Db2 subsystems, RRSAF returns the value `'    '` (four blanks) in all unused array members.

activea
A storage area for an array of 4-byte values into which RRSAF returns an indication of whether a defined subsystem is active. Each value is represented as a fixed 31-bit integer. The value 1 means that the subsystem is active. The value 0 means that the subsystem is not active. The size of this array must be the same as the size of the *ssnma* array. If the array is larger than the number of Db2 subsystems, RRSAF returns the value -1 in all unused array members.

The information in the *activea* array is the information that is available at the point in time that you requested it and might change at any time.

arraysz
A 4-byte area, represented as a fixed 31-bit integer, that specifies the number of entries for the *ssnma* and *activea* arrays. If the number of array entries is insufficient to contain all of the subsystems defined on the current LPAR, RRSAF uses all available entries and returns return code 4.

retcode
A 4-byte area in which RRSAF is to place the return code for this call to the FIND_DB2_SYSTEMS function.

This parameter is optional. If you do not *retcode*, RRSAF places the return code in register 15 and the reason code in register 0.

reascode
A 4-byte area in which RRSAF is to place the reason code for this call to the FIND_DB2_SYSTEMS function.

This parameter is optional. If you do not specify *reascode*, RRSAF places the reason code in register 0.

Example values that the FIND_DB2_SYSTEMS function returns

Assume that two subsystems are defined on the current LPAR. Subsystem DB2A is active, and subsystem DB2B is stopped. Suppose that you invoke RRSAF with the function FIND_DB2_SYSTEMS and a value of 3 for *arraysz*. The *ssnma* array and *activea* array are set to the following values:

Table 34. Example values returned in the ssnma and activeaarrays

Array element number	Values in *ssnma* array	Values in *activea* array
1	DB2A	1
2	DB2B	0
3	(four blanks)	-1

Related tasks
Invoking the Resource Recovery Services attachment facility
The Resource Recovery Services attachment facility (RRSAF) enables your program to communicate with Db2. Invoke RRSAF as an alternative to invoking CAF or when using stored procedures that run in a WLM-established address space. RRSAF has more capabilities than CAF.

RRSAF return codes and reason codes

If you specify return code and reason code parameters in an Resource Recovery Services attachment facility (RRSAF) function call, RRSAF returns the return code and reason code in those parameters. If you do not specify those parameters or implicitly invoke RRSAF, RRSAF puts the return code in register 15 and the reason code in register 0.

When the reason code begins with X'00F3', except for X'00F30006', you can use the RRSAF TRANSLATE function to obtain error message text that can be printed and displayed.

For SQL calls, RRSAF returns standard SQL return codes in the SQLCA. RRSAF returns IFI return codes and reason codes in the instrumentation facility communication area (IFCA).

The following table lists the RRSAF return codes.

Table 35. RRSAF return codes

Return code	Explanation
0	The call completed successfully.
4	Status information is available. See the reason code for details.
>4	The call failed. See the reason code for details.

Related reference

TRANSLATE function for RRSAF

The RRSAF TRANSLATE function converts a hexadecimal reason code for a Db2 error into a signed integer SQL code and a printable error message. The SQL code and message text are placed in the SQLCODE and SQLSTATE host variables or related fields of the SQLCA.

Sample RRSAF scenarios

One or more tasks can use Resource Recovery Services attachment facility (RRSAF) to connect to Db2. This connection can be made either implicitly or explicitly. For explicit connections, a task calls one or more of the RRSAF connection functions.

A single task

The following example pseudocode illustrates a single task running in an address space that explicitly connects to Db2 through RRSAF. z/OS RRS controls commit processing when the task terminates normally.

```
IDENTIFY
SIGNON
CREATE THREAD
SQL or IFI
⋮
TERMINATE IDENTIFY
```

Multiple tasks

In the following scenario, multiple tasks in an address space explicitly connect to Db2 through RRSAF. Task 1 executes no SQL statements and makes no IFI calls. Its purpose is to monitor Db2 termination and startup ECBs and to check the Db2 release level.

```
TASK 1             TASK 2          TASK 3          TASK n

IDENTIFY           IDENTIFY        IDENTIFY        IDENTIFY
                   SIGNON          SIGNON          SIGNON
                   CREATE THREAD   CREATE THREAD   CREATE THREAD
                     SQL             SQL             SQL
                     ...             ...             ...
                     SRRCMIT         SRRCMIT         SRRCMIT
                     SQL             SQL             SQL
                     ...             ...             ...
                     SRRCMIT         SRRCMIT         SRRCMIT
                     ...             ...             ...
TERMINATE IDENTIFY
```

Reusing a Db2 thread

The following example pseudocode shows a Db2 thread that is reused by another user at a point of consistency. When the application calls the SIGNON function for user B, Db2 reuses the plan that is allocated by the CREATE THREAD function for user A.

```
IDENTIFY
SIGNON user A
CREATE THREAD
  SQL
  ...
  SRRCMIT
SIGNON user B
  SQL
  ...
  SRRCMIT
```

Switching Db2 threads between tasks

The following scenario shows how you can switch the threads for four users (A, B, C, and D) among two tasks (1 and 2).

```
Task 1                              Task 2

CTXBEGC (create context a)          CTXBEGC (create context b)
  CTXSWCH(a,0)                        CTXSWCH(b,0)
  IDENTIFY                            IDENTIFY
  SIGNON user A                       SIGNON user B
  CREATE THREAD (Plan A)              CREATE THREAD (plan B)
    SQL                                 SQL
    ...                                 ...
CTXSWCH(0,a)                        CTXSWCH(0,b)


CTXBEGC (create context c)          CTXBEGC (create context d)
  CTXSWCH(c,0)                        CTXSWCH(d,0)
  IDENTIFY                            IDENTIFY
  SIGNON user C                       SIGNON user D
  CREATE THREAD (plan C)              CREATE THREAD (plan D)
    SQL                                 SQL
    ...                                 ...
CTXSWCH(b,c)                        CTXSWCH(0,d)
    SQL (plan B)                        ...
    ...                             CTXSWCH(a,0)
                                        SQL (plan A)
```

The applications perform the following steps:

- Task 1 creates context a, switches contexts so that context a is active for task 1, and calls the IDENTIFY function to initialize a connection to a subsystem. A task must always call the IDENTIFY function before a context switch can occur. After the IDENTIFY operation is complete, task 1 allocates a thread for user A, and performs SQL operations.

  At the same time, task 2 creates context b, switches contexts so that context b is active for task 2, calls the IDENTIFY function to initialize a connection to the subsystem, allocates a thread for user B, and performs SQL operations.

  When the SQL operations complete, both tasks perform RRS context switch operations. Those operations disconnect each Db2 thread from the task under which it was running.
- Task 1 then creates context c, calls the IDENTIFY function to initialize a connection to the subsystem, switches contexts so that context c is active for task 1, allocates a thread for user C, and performs SQL operations for user C.

  Task 2 does the same operations for user D.
- When the SQL operations for user C complete, task 1 performs a context switch operation to perform the following actions:
  - Switch the thread for user C away from task 1.
  - Switch the thread for user B to task 1.

  For a context switch operation to associate a task with a Db2 thread, the Db2 thread must have previously performed an IDENTIFY operation. Therefore, before the thread for user B can be associated with task 1, task 1 must have performed an IDENTIFY operation.
- Task 2 performs two context switch operations to perform the following actions:
  - Disassociate the thread for user D from task 2.
  - Associate the thread for user A with task 2.

Program examples for RRSAF

The Resource Recovery Services attachment facility (RRSAF) enables programs to communicate with Db2. You can use RRSAF as an alternative to CAF.

Example JCL for invoking RRSAF

The following sample JCL shows how to use RRSAF in a batch environment. The DSNRRSAF DD statement starts the RRSAF trace. Use that DD statement only if you are diagnosing a problem.

```
//jobname      JOB      z/OS_jobcard_information
//RRSJCL       EXEC     PGM=RRS_application_program
//STEPLIB      DD       DSN=application_load_library
//             DD       DSN=DB2_load_library

⋮

//SYSPRINT     DD       SYSOUT=*
//DSNRRSAF     DD       DUMMY
//SYSUDUMP     DD       SYSOUT=*
```

Example of loading and deleting the RRSAF language interface

The following code segment shows how an application loads entry points DSNRLI and DSNHLIR of the RRSAF language interface. Storing the entry points in variables LIRLI and LISQL ensures that the application loads the entry points only once. Delete the loaded modules when the application no longer needs to access Db2.

```
****************************** GET LANGUAGE INTERFACE ENTRY ADDRESSES
         LOAD  EP=DSNRLI          Load the RRSAF service request EP
         ST    R0,LIRLI           Save this for RRSAF service requests
         LOAD  EP=DSNHLIR         Load the RRSAF SQL call Entry Point
         ST    R0,LISQL           Save this for SQL calls
*        .
*        .     Insert connection service requests and SQL calls here
*        .
         DELETE EP=DSNRLI         Correctly maintain use count
         DELETE EP=DSNHLIR        Correctly maintain use count
```

Example of using dummy entry point DSNHLI for RRSAF

Each of the Db2 attachment facilities contains an entry point named DSNHLI. When you use RRSAF but do not specify the ATTACH(RRSAF) precompiler option, the precompiler generates BALR instructions to DSNHLI for SQL statements in your program. To find the correct DSNHLI entry point without including DSNRLI in your load module, code a subroutine, with entry point DSNHLI, that passes control to entry point DSNHLIR in the DSNRLI module. DSNHLIR is unique to DSNRLI and is at the same location as DSNHLI in DSNRLI. DSNRLI uses 31-bit addressing. If the application that calls this intermediate subroutine uses 24-bit addressing, the intermediate subroutine must account for the difference.

In the following example, LISQL is addressable because the calling CSECT used the same register 12 as CSECT DSNHLI. Your application must also establish addressability to LISQL.

```
***********************************************************************
* Subroutine DSNHLI intercepts calls to LI EP=DSNHLI
***********************************************************************
         DS    0D
DSNHLI   CSECT                    Begin CSECT
         STM   R14,R12,12(R13)    Prologue
         LA    R15,SAVEHLI        Get save area address
         ST    R13,4(,R15)        Chain the save areas
         ST    R15,8(,R13)        Chain the save areas
         LR    R13,R15            Put save area address in R13
         L     R15,LISQL          Get the address of real DSNHLI
         BASSM R14,R15            Branch to DSNRLI to do an SQL call
*                                 DSNRLI is in 31-bit mode, so use
*                                 BASSM to assure that the addressing
*                                 mode is preserved.
         L     R13,4(,R13)        Restore R13 (caller's save area addr)
```

```
         L     R14,12(,R13)       Restore R14 (return address)
         RETURN (1,12)            Restore R1-12, NOT R0 and R15 (codes)
```

Example of connecting to Db2 with RRSAF

This example uses the variables that are declared in the following code.

```
****************** VARIABLES SET BY APPLICATION ***********************
LIRLI    DS    F                  DSNRLI entry point address
LISQL    DS    F                  DSNHLIR entry point address
SSNM     DS    CL4                DB2 subsystem name for IDENTIFY
CORRID   DS    CL12               Correlation ID for SIGNON
ACCTTKN  DS    CL22               Accounting token for SIGNON
ACCTINT  DS    CL6                Accounting interval for SIGNON
PLAN     DS    CL8                DB2 plan name for CREATE THREAD
COLLID   DS    CL18               Collection ID for CREATE THREAD.  If
*                                 PLAN contains a plan name, not used.
REUSE    DS    CL8                Controls SIGNON after CREATE THREAD
CONTROL  DS    CL8                Action that application takes based
*                                  on return code from RRSAF
****************** VARIABLES SET BY DB2 *******************************
STARTECB DS    F                  DB2 startup ECB
TERMECB  DS    F                  DB2 termination ECB
EIBPTR   DS    F                  Address of environment info block
RIBPTR   DS    F                  Address of release info block
****************************** CONSTANTS *******************************
CONTINUE DC    CL8'CONTINUE'      CONTROL value: Everything OK
IDFYFN   DC    CL18'IDENTIFY          ' Name of RRSAF service
SGNONFN  DC    CL18'SIGNON            ' Name of RRSAF service
CRTHRDFN DC    CL18'CREATE THREAD     ' Name of RRSAF service
TRMTHDFN DC    CL18'TERMINATE THREAD  ' Name of RRSAF service
TMIDFYFN DC    CL18'TERMINATE IDENTIFY' Name of RRSAF service
****************************** SQLCA and RIB **************************
    EXEC SQL INCLUDE SQLCA
         DSNDRIB                  Map the DB2 Release Information Block
******************* Parameter list for RRSAF calls ********************
RRSAFCLL CALL  ,(*,*,*,*,*,*,*,*),VL,MF=L
```

The following example code shows how to issue requests for the RRSAF functions IDENTIFY, SIGNON, CREATE THREAD, TERMINATE THREAD, and TERMINATE IDENTIFY. This example does not show a task that waits on the Db2 termination ECB. You can code such a task and use the z/OS WAIT macro to monitor the ECB. The task that waits on the termination ECB should detach the sample code if the termination ECB is posted. That task can also wait on the Db2 startup ECB. This example waits on the startup ECB at its own task level.

```
***************************** IDENTIFY ********************************
         L     R15,LIRLI          Get the Language Interface address
         CALL  (15),(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB),VL,MF=X
               (E,RRSAFCLL)
         BAL   R14,CHEKCODE       Call a routine (not shown) to check
*                                  return and reason codes
         CLC   CONTROL,CONTINUE   Is everything still OK
         BNE   EXIT               If CONTROL not 'CONTINUE', stop loop
         USING R8,RIB             Prepare to access the RIB
         L     R8,RIBPTR          Access RIB to get DB2 release level
         CLC   RIBREL,RIBR999     DB2 V10 or later?
         BE    USERELX            If RIBREL = '999', use RIBRELX
         WRITE 'The current DB2 release level is' RIBREL
         B     SIGNON             Continue with signon  USERELX
         WRITE 'The current DB2 release level is' RIBRELX
***************************** SIGNON **********************************
SIGNON   L     R15,LIRLI          Get the Language Interface address
         CALL  (15),(SGNONFN,CORRID,ACCTTKN,ACCTINT),VL,MF=(E,RRSAFCLL)
         BAL   R14,CHEKCODE       Check the return and reason codes
*************************** CREATE THREAD *****************************
         L     R15,LIRLI          Get the Language Interface address
         CALL  (15),(CRTHRDFN,PLAN,COLLID,REUSE),VL,MF=(E,RRSAFCLL)
         BAL   R14,CHEKCODE       Check the return and reason codes
****************************** SQL ************************************
*              Insert your SQL calls here.  The DB2 Precompiler
*              generates calls to entry point DSNHLI.  You should
*              code a dummy entry point of that name to intercept
*              all SQL calls.  A dummy DSNHLI is shown in the following
*              section.
************************ TERMINATE THREAD *****************************
         CLC   CONTROL,CONTINUE   Is everything still OK?
```

```
          BNE   EXIT               If CONTROL not 'CONTINUE', shut down
          L     R15,LIRLI          Get the Language Interface address
          CALL  (15),(TRMTHDFN),VL,MF=(E,RRSAFCLL)
          BAL   R14,CHEKCODE       Check the return and reason codes
************************ TERMINATE IDENTIFY **************************
          CLC   CONTROL,CONTINUE   Is everything still OK
          BNE   EXIT               If CONTROL not 'CONTINUE', stop loop
          L     R15,LIRLI          Get the Language Interface address
          CALL  (15),(TMIDFYFN),VL,MF=(E,RRSAFCLL)
          BAL   R14,CHEKCODE       Check the return and reason codes
```

Universal language interface (DSNULI)

The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

The following figure shows the general structure of DSNULI and a program that uses it:

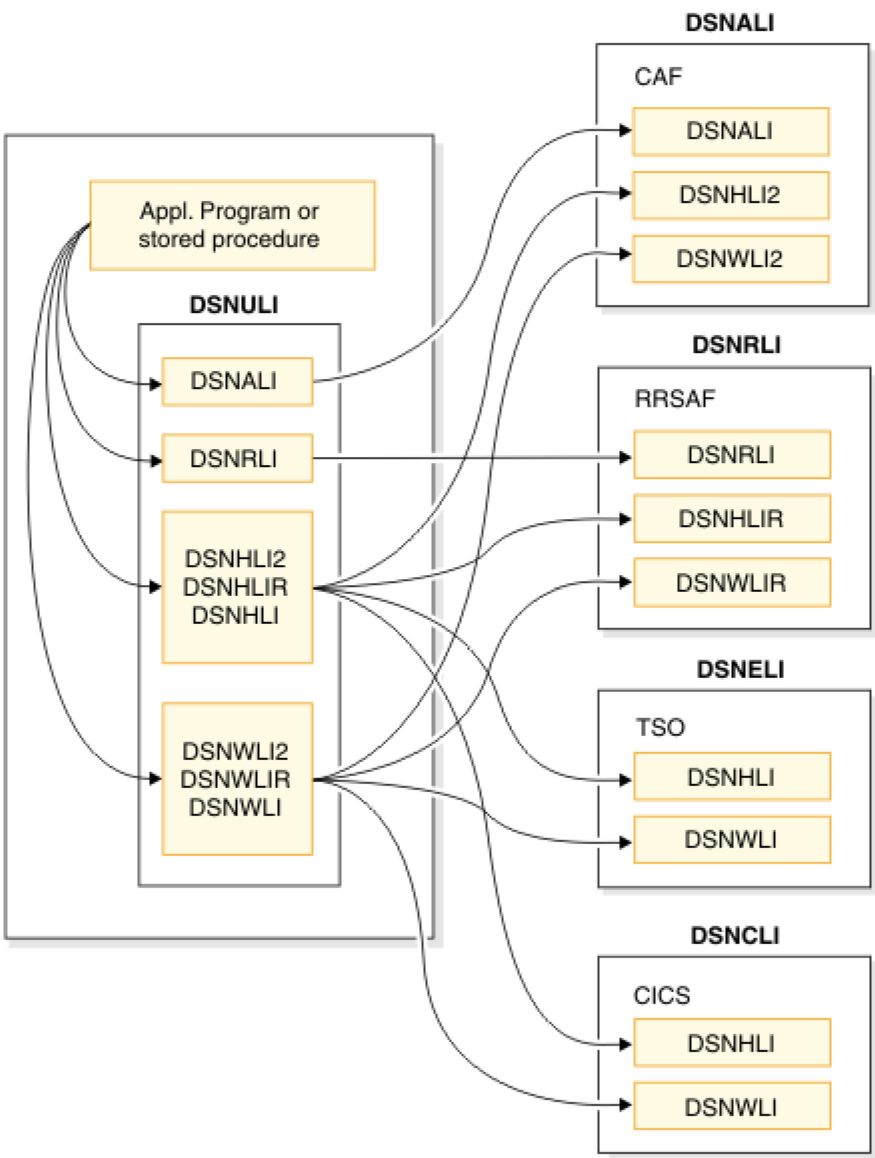


Figure 2. Application program or stored procedure linked with DSNULI

The Db2 load module, DSNULI, is the Universal Language Interface module. DSNULI has no aliases.

DSNULI has the following entry points:

DSNALI
For explicit Db2 Call Attach Facility connection service requests

DSNRLI
For explicit Db2 Resource Recovery Services Attach Facility connection service requests

DSNCLI
For link-editing with CICS

DSNHLI
For generic SQL calls from applications that are designed to run in any environment.

DSNHLI2
For explicit SQL calls by way of the Call Attachment Facility

DSNHLIR
For explicit SQL calls by way of the Resource Recovery Services Attachment Facility

DSNWLI
For generic IFI calls from applications that are designed to run in any environment.

DSNWLI2
For explicit IFI calls by way of the Call Attachment Facility.

DSNWLIR
For explicit IFI call by way of the Resource Recovery Services Attachment Facility

DSNULI dynamically loads and branches to the appropriate language interface module, which is based on the entry point name (for attachment-specific entry points), or based on the current environment (for the generic entry points DSNHLI and DSNWLI).

Related tasks

Link-editing an application with DSNULI
To create a single load module that can be used in more than one attachment environment, you can link-edit your program or stored procedure with the Universal Language Interface module (DSNULI) instead of with one of the environment-specific language interface modules (DSNELI, DSNALI, DSNRLI, or DSNCLI).

Link-editing an application with DSNULI

To create a single load module that can be used in more than one attachment environment, you can link-edit your program or stored procedure with the Universal Language Interface module (DSNULI) instead of with one of the environment-specific language interface modules (DSNELI, DSNALI, DSNRLI, or DSNCLI).

About this task

DSNULI should be link-edited with TSO, CAF, RRSAF applications (including Stored Procedures), and CICS applications. DSNULI does not support dynamic loading of IMS applications. DSNULI determines the run time environment, then dynamically loads and branches to the appropriate language interface module (DSNELI, DSNALI, DSNRLI, or DSNCLI).

The following considerations apply:

- If maximum performance is the primary requirement, link-edit with DSNELI, DSNALI, DSNRLI, or DSNCLI rather than DSNULI. If maintaining a single copy of a load module is the primary requirement, link-edit with DSNULI.
- If CAF implicit connect functionality is required, link-edit your application with DSNALI instead of with DSNULI. DSNULI defaults to RRSAF implicit connections if an attachment environment has not been established upon entry to DSNHLI. Attachment environments are established by calling DSNRLI or DSNALI initially, or by running an SQL application under the TSO command processor or under CICS.
- DSNULI will not explicitly delete the loaded DSNELI, DSNALI, DSNRLI or DSNCLI. If an application cannot tolerate having these modules deleted only at task termination, use DSNELI, DSNALI, DSNRLI or DSNCLI instead of DSNULI.

- DSNULI is shipped with the linkage attributes AMODE(31) and RMODE(ANY) and must be entered in AMODE(31).

Procedure

You can include DSNULI when you link-edit your load module. For example, you can use a linkage editor control statement like this in your JCL:

```
INCLUDE SYSLIB(DSNULI)
```

Results

By coding this statement, you avoid linking to one of the environment-specific language interface modules.

Controlling the CICS attachment facility from an application

Use the CICS attachment facility to access Db2 from CICS application programs.

About this task

You can start and stop the CICS attachment facility from within an application program.

Procedure

To control the CICS attachment facility:

1. To start the CICS attachment facility, perform one of the following actions:
   - Include the following statement in your application:

     ```
     EXEC CICS LINK PROGRAM('DSN2COM0')
     ```

   - Use the system programming interface SET DB2CONN for the CICS Transaction Server.
2. To stop the CICS attachment facility, perform one of the following actions:
   - Include the following statement in your application:

     ```
     EXEC CICS LINK PROGRAM('DSN2COM2')
     ```

   - Use the system programming interface SET DB2CONN for the CICS Transaction Server.

Related information

SET DB2CONN (CICS Transaction Server for z/OS)

Detecting whether the CICS attachment facility is operational

Before you execute SQL statements in a CICS program, you should determine if the CICS attachment facility is available. You do not need to do this test if the CICS attachment facility is started and you are using standby mode.

About this task

When an SQL statement is executed, and the CICS attachment facility is in standby mode, the attachment issues SQLCODE -923 with a reason code that indicates that Db2 is not available.

Procedure

Use the INQUIRE EXITPROGRAM command for the CICS Transaction Server in your application.

The following example shows how to use this command. In this example, the INQUIRE EXITPROGRAM command tests whether the resource manager for SQL, DSNCSQL, is up and running. CICS returns the results in the EIBRESP field of the EXEC interface block (EIB) and in the field whose name is the argument of the CONNECTST parameter (in this case, STST). If the EIBRESP value indicates that the command completed normally and the STST value indicates that the resource manager is available, you can then execute SQL statements.

```
STST     DS    F
ENTNAME  DS    CL8
EXITPROG DS    CL8
⋮
         MVC   ENTNAME,=CL8'DSNCSQL'
         MVC   EXITPROG,=CL8'DSN2EXT1'
         EXEC CICS INQUIRE EXITPROGRAM(EXITPROG)                       X
               ENTRYNAME(ENTNAME) CONNECTST(STST) NOHANDLE
         CLC   EIBRESP,DFHRESP(NORMAL)
         BNE   NOTREADY
         CLC   STST,DFHVALUE(CONNECTED)
         BNE   NOTREADY
UPNREADY DS    0H
         attach is up
NOTREADY DS    0H
         attach is not up yet
```

If you use the INQUIRE EXITPROGRAM command to avoid AEY9 abends and the CICS attachment facility is down, the storm drain effect can occur. The *storm drain effect* is a condition that occurs when a system continues to receive work, even though that system is down.

Related concepts

Storm-drain effect (Db2 Installation and Migration)

Related information

INQUIRE EXITPROGRAM (CICS Transaction Server for z/OS)

-923 (Db2 Codes)

Improving thread reuse in CICS applications

Having transactions reuse threads is generally recommended because each thread creation is associated with a high processor cost.

Procedure

Close all cursors that are declared with the WITH HOLD option before each sync point.

Db2 does not automatically close them. A thread for an application that contains an open cursor cannot be reused. You should close all cursors immediately after you finish using them.

Related concepts

Held and non-held cursors

A held cursor does not close after a commit operation. A cursor that is not held closes after a commit operation. You specify whether you want a cursor to be held or not held by including or omitting the WITH HOLD clause when you declare the cursor.

Chapter 3. Db2 SQL programming

You can use the SQL language to write a statement that describes what you want to do with the data in a database and under what conditions you want to do it.

Structured Query Language (SQL) is a standardized language based on the relational model of data that is used for defining and manipulating data in a relational database. SQL statements can be contained in user-defined functions, user-defined procedures, or triggers, embedded in high-level language programs, dynamically prepared and run, or run interactively.

For information about embedded SQL, see Chapter 4, “Embedded SQL programming,” on page 461.

Creating and modifying Db2 objects from application programs

Your application program can create and manipulate Db2 objects, such as tables, views, triggers, distinct types, user-defined functions, and stored procedures. You must have the appropriate authorizations to create such objects.

Creating tables from application programs

Creating a table provides a logical place to store related data on a Db2 subsystem.

Procedure

Use a CREATE TABLE statement that includes the following elements:

- The name of the table. See Guidelines for table names (Db2 Administration Guide).
- A list of the columns that make up the table. Separate each column description from the next with a comma, and enclose the entire list of column descriptions in parentheses.

  For each column, specify the following information:

  – The name of the column (for example, SERIAL). See Column names (Db2 SQL).
  – The data type and length attribute (for example, CHAR(8)). See Data types of columns (Introduction to Db2 for z/OS).
  – Optionally, specify a default value, or a constraint on the value. You can use the following values:

Keyword	Result
NOT NULL	Specifies the column cannot contain null values
UNIQUE	The value for each row must be unique, and the column cannot contain null values.
DEFAULT	The column has one of the following Db2-assigned default values: - For numeric columns, 0 (zero) is the default value. - For character or graphic fixed-length strings, blank is the default value. - For binary fixed-length strings, a set of hexadecimal zeros is the default value. - For variable-length strings, including LOB strings, the empty string (a string of zero-length) is the default value.

Keyword	Result
	- For datetime columns, the current value of the associated special register is the default value.
DEFAULT *value*	The default value is specified as one of the following values: - A constant - NULL - SESSION_USER, which specifies the value of the SESSION_USER special register at the time when a default value is needed for the column - CURRENT SQLID, which specifies the value of the CURRENT SQLID special register at the time when a default value is needed for the column - The name of a cast function that casts a default value (of a built-in data type) to the distinct type of a column

– Optionally, specify the partitioning method for the data in the table. Db2 uses size-based partitions by default if you do not specify how to partition the data when you create the table. For more information, see Partitioning data in Db2 tables (Db2 Administration Guide).

– Optionally, a referential constraint or check constraint. For more information, see "Check constraints" on page 129 and "Referential constraints" on page 130.

Example

For example, the following SQL statement creates a table named PRODUCT:

```
CREATE TABLE PRODUCT
  (SERIAL       CHAR(8)     NOT NULL,
   DESCRIPTION  VARCHAR(60) DEFAULT,
   MFGCOST      DECIMAL(8,2),
   MFGDEPT      CHAR(3),
   MARKUP       SMALLINT,
   SALESDEPT    CHAR(3),
   CURDATE      DATE        DEFAULT);
```

Related concepts

Db2 tables (Introduction to Db2 for z/OS)

Related tasks

Creating base tables (Db2 Administration Guide)

Related reference

CREATE TABLE (Db2 SQL)

Related information

Lesson 1.2: Creating a table (Introduction to Db2 for z/OS)

Data types of columns

When you create a Db2 table, you define each column to have a specific data type. The data type of a column determines what you can and cannot do with the column.

When you perform operations on columns, the data must be compatible with the data type of the referenced column. For example, you cannot insert character data, such as a last name, into a column whose data type is numeric. Similarly, you cannot compare columns that contain incompatible data types.

The data type for a column can be a distinct type, which is a user-defined data type, or a Db2 built-in data type. As shown in the following figure, Db2 built-in data types have four general categories: datetime, string, numeric, and row identifier (ROWID).

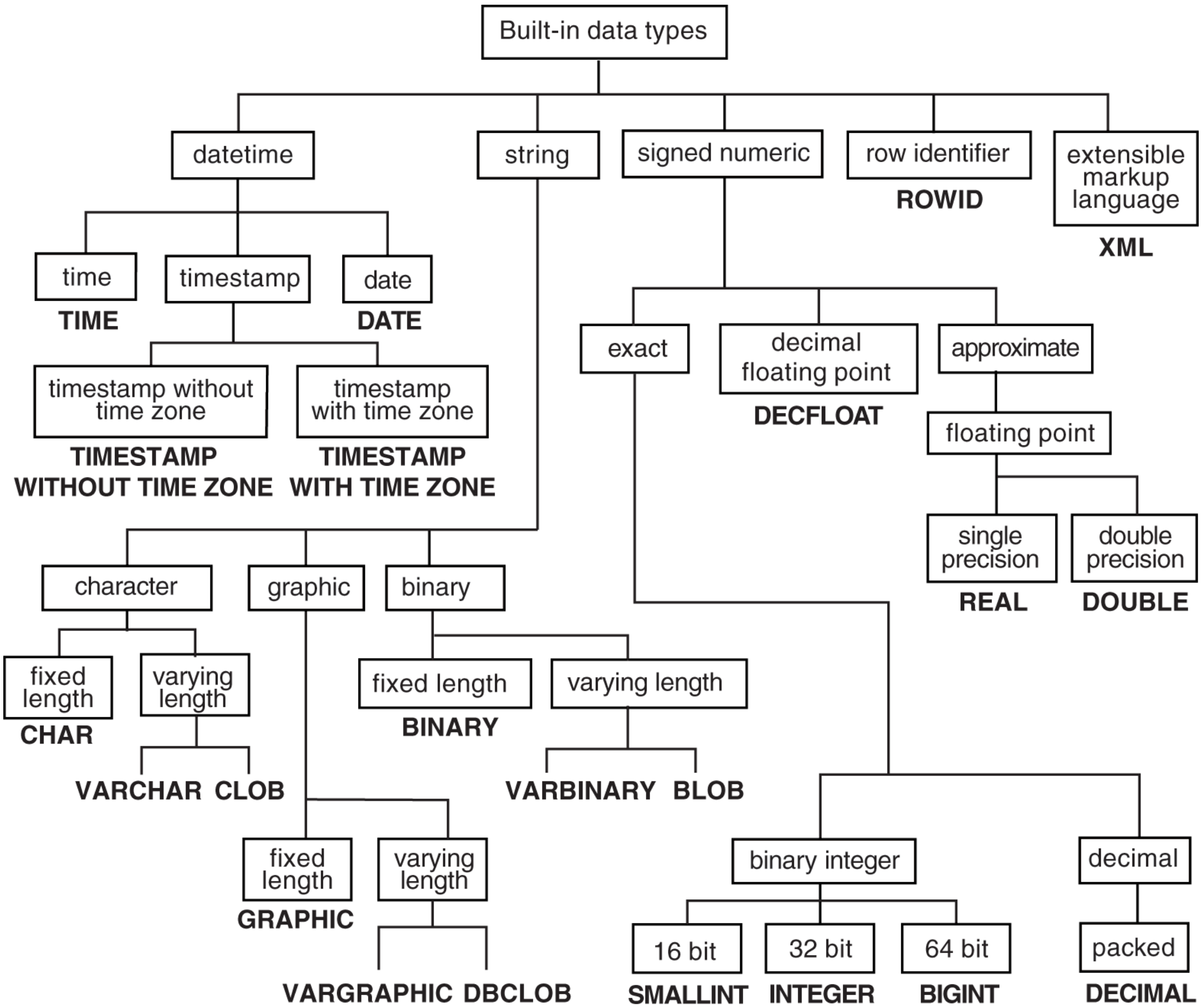


Figure 3. Db2 built-in data types

Related concepts

Assignment and comparison (Db2 SQL)

Casting between data types (Db2 SQL)

Rules for result data types (Db2 SQL)

Distinct types

A *distinct type* is a user-defined data type that shares its internal representation with a built-in data type (its *source type*), but is considered to be a separate and incompatible data type for most operations.

Data types (Db2 SQL)

Storing LOB data in a table

Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

About this task

Large object and *LOB* refer to Db2 objects that you can use to store large amounts of data. A LOB is a varying-length character string that can contain up to 2 GB - 1 of data. Db2 supports the following LOB data types:

Binary large object (BLOB)
Use a BLOB to store binary data such as pictures, voice, and mixed media.

Character large object (CLOB)
Use a CLOB to store SBCS or mixed character data, such as documents.

Double-byte character large object (DBCLOB)
Use a DBCLOB to store data that consists of only DBCS data.

For more information about LOB data types, see Large objects (LOBs) (Db2 SQL).

You can use Db2 to store LOB data, but this data is stored differently than other kinds of data.

Although a table can have a LOB column, the actual LOB data is stored in a another table, which called the auxiliary table. This auxiliary table exists in a separate table space called a LOB table space. One auxiliary table must exist for each LOB column. The table with the LOB column is called the base table. The base table has a ROWID column that Db2 uses to locate the data in the auxiliary table. The auxiliary table must have exactly one index.

Procedure

To store LOB data in Db2:

1. Define one or more columns of the appropriate LOB type and optionally a row identifier (ROWID) column by executing a CREATE TABLE statement or one or more ALTER TABLE statements.

   Define only one ROWID column, even if the table is to have multiple LOB columns. If you do not create a ROWID column before you define a LOB column, Db2 implicitly creates a ROWID column with the IMPLICITLY HIDDEN attribute and appends it as the last column of the table.

   If you add a ROWID column after you add a LOB column, the table has two ROWID columns: the implicitly-created column and the explicitly-created column. In this case, Db2 ensures that the values of the two ROWID columns are always identical.

   If Db2 implicitly creates the table space for this table or CURRENT RULES is set to STD, Db2 creates the necessary auxiliary objects for you and you can skip steps 2 and 3.
2. If you explicitly created the table space for this table and the CURRENT RULES special register is not set to STD, create a LOB table space and auxiliary table by using the CREATE LOB TABLESPACE and CREATE AUXILIARY TABLE statements.
   - If your base table is nonpartitioned, create one LOB table space and for each column create one auxiliary table.
   - If your base table is partitioned, create one LOB table space for each partition and one auxiliary table for each column. For example, if your base table has three partitions, you must create three LOB table spaces and three auxiliary tables for each LOB column.
3. If you explicitly created the table space for this table and the CURRENT RULES special register is not set to STD, create one index for each auxiliary table by using the CREATE INDEX statement.
4. Insert the LOB data into Db2 by using one of the following techniques:
   - If the total length of a LOB column and the base table row is less than 32 KB, use the LOAD utility and specify the base table.

- Otherwise, use INSERT, UPDATE, or MERGE statements and specify the base table. If you use the INSERT statement, ensure that you application has enough storage available to hold the entire value that is to be put into the LOB column.

Example

Suppose that you want to add a resume for each employee to the employee table. The employee resumes are no more than 5 MB in size. Because the employee resumes contain single-byte characters, you can define the resumes to Db2 as CLOBs. You therefore need to add a column of data type CLOB with a length of 5 MB to the employee table. If you want to define a ROWID column explicitly, you must define it before you define the CLOB column.

First, execute an ALTER TABLE statement to add the ROWID column, and then execute another ALTER TABLE statement to add the CLOB column. The following statements create these columns:

```
ALTER TABLE EMP
  ADD ROW_ID ROWID NOT NULL GENERATED ALWAYS;
COMMIT;
ALTER TABLE EMP
  ADD EMP_RESUME CLOB(5M);
COMMIT;
```

If you explicitly created the table space for this table and the CURRENT RULES special register is not set to STD, you then need to define a LOB table space and an auxiliary table to hold the employee resumes. You also need to define an index on the auxiliary table. You must define the LOB table space in the same database as the associated base table. The following statements create these objects:

```
CREATE LOB TABLESPACE RESUMETS
  IN DSN8D11A
  LOG NO
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN DSN8D11A.RESUMETS
  STORES DSN8B10.EMP
  COLUMN EMP_RESUME;
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```

You can then load your employee resumes into Db2. In your application, you can define a host variable to hold the resume, copy the resume data from a file into the host variable, and then execute an UPDATE statement to copy the data into Db2. Although the LOB data is stored in the auxiliary table, your UPDATE statement specifies the name of the base table. The following code declares a host variable to store the resume in the C language:

```
SQL TYPE is CLOB (5M) resumedata;
```

The following UPDATE statement copies the data into Db2:

```
UPDATE EMP SET EMP_RESUME=:resumedata
  WHERE EMPNO=:employeenum;
```

In this statement, `employeenum` is a host variable that identifies the employee who is associated with a resume.

Related concepts

Large objects (LOBs) (Db2 SQL)

Creation of large objects (Introduction to Db2 for z/OS)

Related reference

CREATE TABLE (Db2 SQL)

CREATE AUXILIARY TABLE (Db2 SQL)

CREATE LOB TABLESPACE (Db2 SQL)

CREATE INDEX (Db2 SQL)

Identity columns

An identity column contains a unique numeric value for each row in the table. Db2 can automatically generate sequential numeric values for this column as rows are inserted into the table. Thus, identity columns are ideal for primary key values, such as employee numbers or product numbers.

Using identity columns as keys

If you define a column with the AS IDENTITY attribute, and with the GENERATED ALWAYS and NO CYCLE attributes, Db2 automatically generates a monotonically increasing or decreasing sequential number for the value of that column when a new row is inserted into the table. However, for Db2 to guarantee that the values of the identity column are unique, you should define a unique index on that column.

You can use identity columns for primary keys that are typically unique sequential numbers, for example, order numbers or employee numbers. By doing so, you can avoid the concurrency problems that can result when an application generates its own unique counter outside the database.

Recommendation: Set the values of the foreign keys in the dependent tables after loading the parent table. If you use an identity column as a parent key in a referential integrity structure, loading data into that structure could be quite complicated. The values for the identity column are not known until the table is loaded because the column is defined as GENERATED ALWAYS.

You might have gaps in identity column values for the following reasons:

- If other applications are inserting values into the same identity column
- If Db2 terminates abnormally before it assigns all the cached values
- If your application rolls back a transaction that inserts identity values

Defining an identity column

You can define an identity column as either GENERATED BY DEFAULT or GENERATED ALWAYS:

- If you define the column as GENERATED BY DEFAULT, you can insert a value, and Db2 provides a default value if you do not supply one.
- If you define the column as GENERATED ALWAYS, Db2 always generates a value for the column, and you cannot insert data into that column. If you want the values to be unique, you must define the identity column with GENERATED ALWAYS and NO CYCLE and define a unique index on that column.

The values that Db2 generates for an identity column depend on how the column is defined. The START WITH option determines the first value that Db2 generates. The values advance by the INCREMENT BY value in ascending or descending order.

The MINVALUE and MAXVALUE options determine the minimum and maximum values that Db2 generates. However, the The CYCLE or NO CYCLE option determines whether Db2 wraps values when it has generated all values between the START WITH value and MAXVALUE if the values are ascending, or between the START WITH value and MINVALUE if the values are descending. MINVALUE and MAXVALUE do not constrain a START WITH or RESTART WITH value.

Example: Using GENERATED ALWAYS and CYCLE

Suppose that table T1 is defined with GENERATED ALWAYS and CYCLE:

```
CREATE TABLE T1
  (CHARCOL1 CHAR(1),
   IDENTCOL1 SMALLINT GENERATED ALWAYS AS IDENTITY
     (START WITH -1,
      INCREMENT BY 1,
      CYCLE,
      MINVALUE -3,
      MAXVALUE 3));
```

Now suppose that you execute the following INSERT statement eight times:

```
INSERT INTO T1 (CHARCOL1) VALUES ('A');
```

When Db2 generates values for IDENTCOL1, it starts with -1 and increments by 1 until it reaches the MAXVALUE of 3 on the fifth INSERT. To generate the value for the sixth INSERT, Db2 cycles back to MINVALUE, which is -3. T1 looks like this after the eight INSERT statements are executed:

```
CHARCOL1  IDENTCOL1
========  =========
A                -1
A                 0
A                 1
A                 2
A                 3
A                -3
A                -2
A                -1
```

The value of IDENTCOL1 for the eighth INSERT repeats the value of IDENTCOL1 for the first INSERT.

Example: START WITH or RESTART WITH values outside the range for cycling

The MINVALUE and MAXVALUE options do not constrain the START WITH value. That is, the START WITH clause can be used to start the generation of values outside the range that is used for cycles. However, the next generated value after the specified START WITH value is MNVALUE for an ascending identity column or MAXVALUE for a descending identity column. The same is true if you alter the identity column and specify a RESTART WITH value.

Consider T1 from the previous example, and suppose that you alter the table with a statement that specifies the following keywords.

```
ALTER TABLE T1
  ALTER COLUMN IDENTCOL1 SET GENERATED ALWAYS RESTART WITH 99;
```

Now suppose that you execute the following INSERT statement three times:

```
INSERT INTO T1 (CHARCOL1) VALUES ('B');
```

When Db2 generates the IDENTCOL1 value, it starts with 99. However, for the next generated value, Db2 again cycles back to MINVALUE, which is -3. T1 looks like this after the three INSERT statements are executed:

```
CHARCOL1  IDENTCOL1
========  =========
A                -1
A                 0
A                 1
A                 2
A                 3
A                -3
A                -2
A                -1
B                99
B                -3
B                -2
```

Identity columns as primary keys

The SELECT from INSERT statement enables you to insert a row into a parent table with its primary key defined as a Db2-generated identity column, and retrieve the value of the primary or parent key. You can then use this generated value as a foreign key in a dependent table.

In addition, you can use the IDENTITY_VAL_LOCAL function to return the most recently assigned value for an identity column.

Example: Using SELECT from INSERT

Suppose that an EMPLOYEE table and a DEPARTMENT table are defined in the following way:

```
CREATE TABLE EMPLOYEE
  (EMPNO      INTEGER GENERATED ALWAYS AS IDENTITY
              PRIMARY KEY NOT NULL,
   NAME       CHAR(30) NOT NULL,
   SALARY     DECIMAL(7,2) NOT NULL,
   WORKDEPT   SMALLINT);

CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT NOT NULL PRIMARY KEY,
   DEPTNAME   VARCHAR(30),
   MGRNO      INTEGER NOT NULL,
   CONSTRAINT REF_EMPNO FOREIGN KEY (MGRNO)
     REFERENCES EMPLOYEE (EMPNO) ON DELETE RESTRICT);

ALTER TABLE EMPLOYEE ADD
  CONSTRAINT REF_DEPTNO FOREIGN KEY (WORKDEPT)
     REFERENCES DEPARTMENT (DEPTNO) ON DELETE SET NULL;
```

When you insert a new employee into the EMPLOYEE table, to retrieve the value for the EMPNO column, you can use the following SELECT from INSERT statement:

```
EXEC SQL
  SELECT EMPNO INTO :hv_empno
    FROM FINAL TABLE (INSERT INTO EMPLOYEE (NAME, SALARY, WORKDEPT)
                      VALUES ('New Employee', 75000.00, 11));
```

The SELECT statement returns the Db2-generated identity value for the EMPNO column in the host variable :hv_empno.

You can then use the value in :hv_empno to update the MGRNO column in the DEPARTMENT table with the new employee as the department manager:

```
EXEC SQL
  UPDATE DEPARTMENT
    SET MGRNO = :hv_empno
    WHERE DEPTNO = 11;
```

Related concepts

Rules for inserting data into an identity column
An *identity column* contains a unique numeric value for each row in the table. Whether you can insert data into an identity column and how that data gets inserted depends on how the column is defined.

Related tasks

Selecting values while inserting data
When you insert rows into a table, you can also select values from the inserted rows at the same time.

Related reference

IDENTITY_VAL_LOCAL (Db2 SQL)

Creating tables for data integrity

To ensure that only valid data is added to your tables, you can use constraints, triggers, and unique indexes. For example, you might need to ensure that all items in your inventory table have valid item numbers and to prevent items without valid item numbers from being added.

About this task

Introductory concepts

Creation of relationships with referential constraints (Introduction to Db2 for z/OS)

Related concepts

Creation of relationships with referential constraints (Introduction to Db2 for z/OS)

Related tasks

Altering a table for referential integrity (Db2 Administration Guide)

Creating indexes to improve referential integrity performance for foreign keys (Db2 Performance)

Creating tables for data integrity
To ensure that only valid data is added to your tables, you can use constraints, triggers, and unique indexes. For example, you might need to ensure that all items in your inventory table have valid item numbers and to prevent items without valid item numbers from being added.

Using referential integrity for data consistency (Managing Security)

Ways to maintain data integrity

When you add or modify data in a Db2 table, you need to ensure that the data is valid. Two techniques that you can use to ensure valid data are constraints and triggers.

Constraints are rules that limit the values that you can insert, delete, or update in a table. There are two types of constraints:

- Check constraints determine the values that a column can contain. Check constraints are discussed in "Check constraints" on page 129.
- Referential constraints preserve relationships between tables. Referential constraints are discussed in "Referential constraints" on page 130. A specific type of referential constraints, the informational referential constraint, is discussed in "Informational referential constraints" on page 132.

To maintain data integrity Db2 enforces check constraints and referential constraints on data in a table. When these types of constraints are violated or might be violated, Db2 places the table space or partition that contains the table in CHECK-pending status.

Triggers are a series of actions that are invoked when a table is updated. Triggers are discussed in "Creating a trigger" on page 151.

Related reference

CHECK-pending status (Db2 Utilities)

Check constraints

A *check constraint* is a rule that specifies the values that are allowed in one or more columns of every row of a base table. For example, you can define a check constraint to ensure that all values in a column that contains ages are positive numbers.

Check constraints designate the values that specific columns of a base table can contain, providing you a method of controlling the integrity of data entered into tables. You can create tables with check constraints using the CREATE TABLE statement, or you can add the constraints with the ALTER TABLE statement. However, if the check integrity is compromised or cannot be guaranteed for a table, the table space or partition that contains the table is placed in a check pending state. Check integrity is the condition that exists when each row of a table conforms to the check constraints defined on that table.

For example, you might want to make sure that no salary can be below 15000 dollars. To do this, you can create the following check constraint:

```
CREATE TABLE EMPSAL
(ID          INTEGER      NOT NULL,
 SALARY      INTEGER      CHECK (SALARY >= 15000));
```

Using check constraints makes your programming task easier, because you do not need to enforce those constraints within application programs or with a validation routine. Define check constraints on one or more columns in a table when that table is created or altered.

Check constraint considerations

The syntax of a check constraint is checked when the constraint is defined, but the meaning of the constraint is not checked. The following examples show mistakes that are not caught. Column C1 is defined as INTEGER NOT NULL.

Allowable but mistaken check constraints:

- A self-contradictory check constraint:

```
CHECK (C1 > 5 AND C1 < 2)
```

- Two check constraints that contradict each other:

```
CHECK (C1 > 5)
CHECK (C1 < 2)
```

- Two check constraints, one of which is redundant:

```
CHECK (C1 >  0)
CHECK (C1 >= 1)
```

- A check constraint that contradicts the column definition:

```
CHECK (C1 IS NULL)
```

- A check constraint that repeats the column definition:

```
CHECK (C1 IS NOT NULL)
```

A check constraint is not checked for consistency with other types of constraints. For example, a column in a dependent table can have a referential constraint with a delete rule of SET NULL. You can also define a check constraint that prohibits nulls in the column. As a result, an attempt to delete a parent row fails, because setting the dependent row to null violates the check constraint.

Similarly, a check constraint is not checked for consistency with a validation routine, which is applied to a table before a check constraint. If the routine requires a column to be greater than or equal to 10 and a check constraint requires the same column to be less than 10, table inserts are not possible. Plans and packages do not need to be rebound after check constraints are defined on or removed from a table.

When check constraints are enforced

After check constraints are defined on a table, any change must satisfy those constraints if it is made by:

- The LOAD utility with the option ENFORCE CONSTRAINT
- An SQL insert operation
- An SQL update operation

A row satisfies a check constraint if its condition evaluates either to true or to unknown. A condition can evaluate to unknown for a row if one of the named columns contains the null value for that row.

Any constraint defined on columns of a base table applies to the views defined on that base table.

When you use ALTER TABLE to add a check constraint to already populated tables, the enforcement of the check constraint is determined by the value of the CURRENT RULES special register as follows:

- If the value is STD, the check constraint is enforced immediately when it is defined. If a row does not conform, the check constraint is not added to the table and an error occurs.
- If the value is Db2, the check constraint is added to the table description but its enforcement is deferred. Because there might be rows in the table that violate the check constraint, the table is placed in CHECK-pending status.

Referential constraints
A *referential constraint* is a rule that specifies that the only valid values for a particular column are those values that exist in another specified table column. For example, a referential constraint can ensure that all customer IDs in a transaction table exist in the ID column of a customer table.

A table can serve as the "master list" of all occurrences of an entity. In the sample application, the employee table serves that purpose for employees; the numbers that appear in that table are the only valid employee numbers. Likewise, the department table provides a master list of all valid department numbers; the project activity table provides a master list of activities performed for projects; and so on.

The following figure shows the relationships that exist among the tables in the sample application. Arrows point from parent tables to dependent tables.

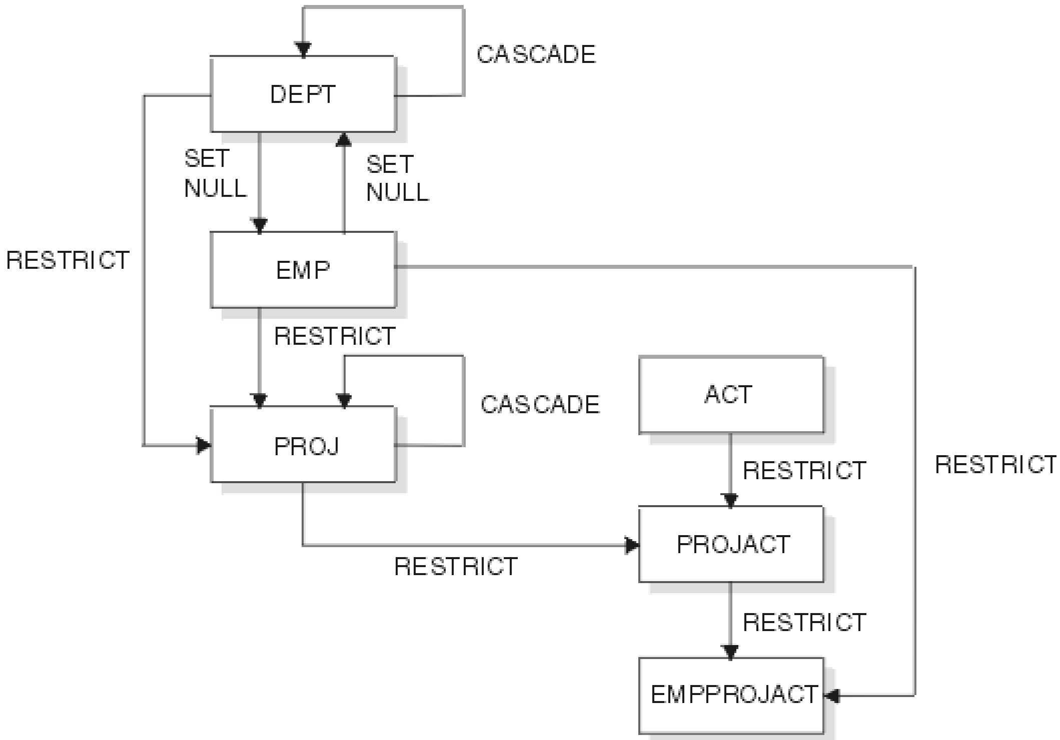


Figure 4. Relationships among tables in the sample application

When a table refers to an entity for which there is a master list, it should identify an occurrence of the entity that actually appears in the master list; otherwise, either the reference is invalid or the master list is incomplete. Referential constraints enforce the relationship between a table and a master list.

Restrictions on cycles of dependent tables
A *cycle* is a set of two or more tables. The tables are ordered so that each is a dependent of the one before it, and the first is a dependent of the last. Every table in the cycle is a descendent of itself. Db2 restricts certain operations on cycles.

In the sample application, the employee and department tables are a cycle; each is a dependent of the other.

Db2 does not allow you to create a cycle in which a delete operation on a table involves that same table. Enforcing that principle creates rules about adding a foreign key to a table:

- In a cycle of two tables, neither delete rule can be CASCADE.
- In a cycle of more than two tables, two or more delete rules must not be CASCADE. For example, in a cycle with three tables, two of the delete rules must be other than CASCADE. This concept is illustrated in The following figure. The cycle on the left is valid because two or more of the delete rules are not CASCADE. The cycle on the right is invalid because it contains two cascading deletes.

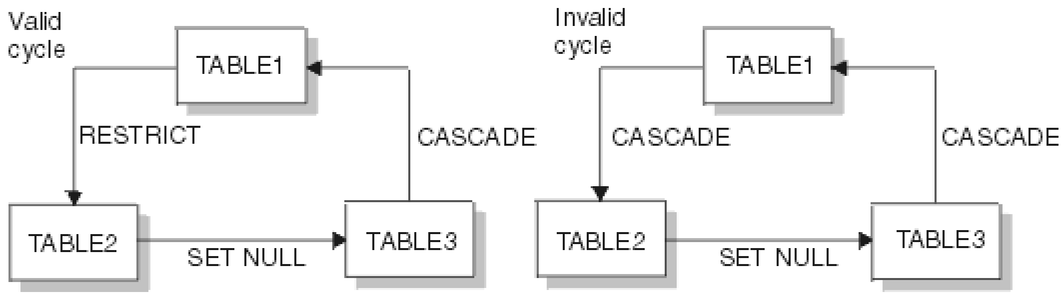


Figure 5. Valid and invalid delete cycles

Alternatively, a delete operation on a self-referencing table must involve the same table, and the delete rule there must be CASCADE or NO ACTION.

Recommendation: Avoid creating a cycle in which all the delete rules are RESTRICT and none of the foreign keys allows nulls. If you do this, no row of any of the tables can ever be deleted.

Referential constraints on tables with multilevel security with row-level granularity
You cannot use referential constraints on a security label column, which is used for multilevel security with row-level granularity. However, you can use referential constraints on other columns in the row.

Db2 does not enforce multilevel security with row-level granularity when it is already enforcing referential constraints. Referential constraints are enforced when the following situations occur:

- An insert operation is applied to a dependent table.
- An update operation is applied to a foreign key of a dependent table, or to the parent key of a parent table.
- A delete operation is applied to a parent table. In addition to all referential constraints being enforced, the Db2 system enforces all delete rules for all dependent rows that are affected by the delete operation. If all referential constraints and delete rules are not satisfied, the delete operation will not succeed.
- The LOAD utility with the ENFORCE CONSTRAINTS option is run on a dependent table.
- The CHECK DATA utility is run.

Related concepts
Multilevel security (Managing Security)

Informational referential constraints
An informational referential constraint is a referential constraint that Db2 does not enforce during normal operations. Use these constraints only when referential integrity can be enforced by another means, such as when retrieving data from other sources. These constraints might improve performance by enabling the query to qualify for automatic query rewrite.

Db2 ignores informational referential constraints during insert, update, and delete operations. Some utilities ignore these constraints; other utilities recognize them. For example, CHECK DATA and LOAD ignore these constraints. QUIESCE TABLESPACESET recognizes these constraints by quiescing all table spaces related to the specified table space.

You should use this type of referential constraint only when an application process verifies the data in a referential integrity relationship. For example, when inserting a row in a dependent table, the application should verify that a foreign key exists as a primary or unique key in the parent table. To define an informational referential constraint, use the NOT ENFORCED option of the referential constraint definition in a CREATE TABLE or ALTER TABLE statement.

Informational referential constraints are often useful, especially in a data warehouse environment, for several reasons:

- To avoid the overhead of enforcement by Db2.

Typically, data in a data warehouse has been extracted and cleansed from other sources. Referential integrity might already be guaranteed. In this situation, enforcement by Db2 is unnecessary.

- To allow more queries to qualify for automatic query rewrite.

  Automatic query rewrite is a process that examines a submitted query that references source tables and, if appropriate, rewrites the query so that it executes against a materialized query table that has been derived from those source tables. This process uses informational referential constraints to determine whether the query can use a materialized query table. Automatic query rewrite results in a significant reduction in query run time, especially for decision-support queries that operate over huge amounts of data.

Related tasks

Using materialized query tables to improve SQL performance (Db2 Performance)

Related reference

CREATE TABLE (Db2 SQL)

Defining a parent key and unique index

A *parent key* is either a primary key or a unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the constraint. You must create a unique index on a parent key.

About this task

The primary key of a table, if one exists, uniquely identifies each occurrence of an entity in the table. The PRIMARY KEY clause of the CREATE TABLE or ALTER TABLE statements identifies the column or columns of the primary key. Each identified column must be defined as NOT NULL.

Another way to allow only unique values in a column is to specify the UNIQUE clause when you create or alter a table.

A table that is to be a parent of dependent tables must have a primary or a unique key; the foreign keys of the dependent tables refer to the primary or unique key. Otherwise, a primary key is optional. Consider defining a primary key if each row of your table does pertain to a unique occurrence of some entity. If you define a primary key, an index must be created (the *primary index*) on the same set of columns, in the same order as those columns. If you are defining referential constraints for Db2 to enforce, takes steps to maintain data integrity read before creating or altering any of the tables involved.

A table can have no more than one primary key. A primary key has the same restrictions as index keys:

- The key can include no more than 64 columns.
- You cannot specify a column name twice.
- The sum of the column length attributes cannot be greater than 2000.

You define a list of columns as the primary key of a table with the PRIMARY KEY clause in the CREATE TABLE statement.

Procedure

Use the PRIMARY KEY clause in an ALTER TABLE statement. In this case, a unique index must already exist.

Consider the following items when you plan for primary keys:

- The theoretical model of a relational database suggests that every table should have a primary key to uniquely identify the entities it describes. However, you must weigh that model against the potential cost of index maintenance overhead. Db2 does not require you to define a primary key for tables with no dependents.
- Choose a primary key whose values will not change over time. Choosing a primary key with persistent values enforces the good practice of having unique identifiers that remain the same for the lifetime of the entity occurrence.

- A primary key column should not have default values unless the primary key is a single TIMESTAMP column.
- Choose the minimum number of columns to ensure uniqueness of the primary key.
- A view that can be updated that is defined on a table with a primary key should include all columns of the key. Although this is necessary only if the view is used for inserts, the unique identification of rows can be useful if the view is used for updates, deletes, or selects.
- Drop a primary key later if you change your database or application using SQL.

Related concepts

Ways to maintain data integrity
When you add or modify data in a Db2 table, you need to ensure that the data is valid. Two techniques that you can use to ensure valid data are constraints and triggers.

Related reference

ALTER TABLE (Db2 SQL)

CREATE TABLE (Db2 SQL)

Parent key columns
A *parent key* is either a primary key or a unique key in the parent table of a referential constraint. This key consists of a column or set of columns. The values of a parent key determine the valid values of the foreign key in the constraint.

If every row in a table represents relationships for a unique entity, the table should have one column or a set of columns that provides a unique identifier for the rows of the table. This column (or set of columns) is called the *parent key* of the table. To ensure that the parent key does not contain duplicate values, you must create a unique index on the column or columns that constitute the parent key. Defining the parent key is called entity integrity, because it requires each entity to have a unique key.

In some cases, using a timestamp as part of the key can be helpful, for example when a table does not have a "natural" unique key or if arrival sequence is the key.

Primary keys for some of the sample tables are:

Table
: **Key Column**

Employee table
: EMPNO

Department table
: DEPTNO

Project table
: PROJNO

Table 36 on page 134 shows part of the project table which has the primary key column, PROJNO.

Table 36. Part of the project table with the primary key column, PROJNO

PROJNO	PROJNAME	DEPTNO
MA2100	WELD LINE AUTOMATION	D01
MA2110	W L PROGRAMMING	D11

Table 37 on page 134 shows part of the project activity table, which has a primary key that contains more than one column. The primary key is a *composite key*, which consists of the PRONNO, ACTNO, and ACSTDATE columns.

Table 37. Part of the Project activities table with a composite primary key

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	.50	1982-01-01	1982-07-01

Table 37. Part of the Project activities table with a composite primary key (continued)				
PROJNO	**ACTNO**	**ACSTAFF**	**ACSTDATE**	**ACENDATE**
AD3110	10	1.00	1982-01-01	1983-01-01
AD3111	60	.50	1982-03-15	1982-04-15

Defining a foreign key

Use foreign keys to enforce referential relationships between tables. A *foreign key* is a column or set of columns that references the parent key in the parent table.

Before you begin

The following prerequisites are met:

- The privilege set must include the ALTER or the REFERENCES privilege on the columns of the parent key.
- A unique index exists on the parent key columns of the parent table.

Procedure

To define a foreign key, use one of the following approaches:

- Issue a CREATE TABLE statement and specify a FOREIGN KEY clause.
  a) Choose a constraint name for the relationship that is defined by a foreign key.
     If you do not choose a name, Db2 generates one from the name of the first column of the foreign key, in the same way that it generates the name of an implicitly created table space.
     For example, the names of the relationships in which the employee-to-project activity table is a dependent would, by default, be recorded (in column RELNAME of SYSIBM.SYSFOREIGNKEYS) as EMPNO and PROJNO.
     The name is used in error messages, queries to the catalog, and DROP FOREIGN KEY statements. Hence, you might want to choose one if you are experimenting with your database design and have more than one foreign key that begins with the same column (otherwise Db2 generates the name).
  b) Specify column names that identify the columns of the parent key.
     A foreign key can refer to either a unique or a primary key of the parent table. If the foreign key refers to a non-primary unique key, you must specify the column names of the key explicitly. If the column names of the key are not specified explicitly, the default is to refer to the column names of the primary key of the parent table.
- Issue an ALTER TABLE statement and specify the FOREIGN KEY clause.
  You can add a foreign key to an existing table; in fact, that is sometimes the only way to proceed. To make a table self-referencing, you must add a foreign key after creating it. When a foreign key is added to a populated table, the table space is put into CHECK-pending status.

Example

The following example shows a CREATE TABLE statement that specifies constraint names REPAPA and REPAE for the foreign keys in the employee-to-project activity table.

```
CREATE TABLE DSN8B10.EMPPROJACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       CONSTRAINT REPAPA FOREIGN KEY (PROJNO, ACTNO)
               REFERENCES DSN8B10.PROJACT ON DELETE RESTRICT,
       CONSTRAINT REPAE FOREIGN KEY (EMPNO)
               REFERENCES DSN8B10.EMP ON DELETE RESTRICT)
  IN DATABASE DSN8D11A;
```

What to do next

If rows of the parent table are often deleted, it is best to create an index on the foreign key.

Related tasks

Adding parent keys and foreign keys (Db2 Administration Guide)

Related reference

CREATE TABLE (Db2 SQL)

ALTER TABLE (Db2 SQL)

SYSFOREIGNKEYS catalog table (Db2 SQL)

Maintaining referential integrity when using data encryption

If you use encrypted data in a referential constraint, the primary key of the parent table and the foreign key of the dependent table must have the same encrypted value.

About this task

The encrypted value should be extracted from the parent table (the primary key) and used for the dependent table (the foreign key). You can do this in one of the following two ways:

- Use the FINAL TABLE clause on a SELECT from UPDATE, SELECT from INSERT, or SELECT from MERGE statement.
- Use the ENCRYPT_TDES function to encrypt the foreign key using the same password as the primary key. The encrypted value of the foreign key will be the same as the encrypted value of the primary key.

The SET ENCRYPTION PASSWORD statement sets the password that will be used for the ENCRYPT_TDES function.

Related reference

ENCRYPT_TDES (Db2 SQL)

ENCRYPTION PASSWORD (Db2 SQL)

Creating work tables for the EMP and DEPT sample tables

Before testing SQL statements that insert, update, and delete rows in the DSN8B10.EMP and DSN8B10.DEPT sample tables, you should create duplicates of these tables. Create duplicates so that the original sample tables remain intact. These duplicate tables are called *work tables*.

About this task

This topic shows how to create the department and employee work tables and how to fill a work table with the contents of another table:

Each of these topics assumes that you logged on by using your own authorization ID. The authorization ID qualifies the name of each object that you create. For example, if your authorization ID is SMITH, and you create table YDEPT, the name of the table is SMITH.YDEPT. If you want to access table DSN8B10.DEPT, you must refer to it by its complete name. If you want to access your own table YDEPT, you need only to refer to it as YDEPT.

Use the following statements to create a new department table called YDEPT, modeled after the existing table, DSN8B10.DEPT, and an index for YDEPT:

```
CREATE TABLE YDEPT
  LIKE DSN8B10.DEPT;
```

```
CREATE UNIQUE INDEX YDEPTX
  ON YDEPT (DEPTNO);
```

If you want DEPTNO to be a primary key, as in the sample table, explicitly define the key. Use an ALTER TABLE statement, as in the following example:

```
ALTER TABLE YDEPT
  PRIMARY KEY(DEPTNO);
```

You can use an INSERT statement to copy the rows of the result table of a fullselect from one table to another. The following statement copies all of the rows from DSN8B10.DEPT to your own YDEPT work table:

```
INSERT INTO YDEPT
  SELECT *
    FROM DSN8B10.DEPT;
```

For information about using the INSERT statement, see "Inserting rows by using the INSERT statement" on page 335.

You can use the following statements to create a new employee table called YEMP:

```
CREATE TABLE YEMP
      (EMPNO     CHAR(6)        PRIMARY KEY NOT NULL,
       FIRSTNME  VARCHAR(12)    NOT NULL,
       MIDINIT   CHAR(1)        NOT NULL,
       LASTNAME  VARCHAR(15)    NOT NULL,
       WORKDEPT  CHAR(3)        REFERENCES YDEPT
                                  ON DELETE SET NULL,
       PHONENO   CHAR(4)        UNIQUE NOT NULL,
       HIREDATE  DATE                   ,
       JOB       CHAR(8)                ,
       EDLEVEL   SMALLINT               ,
       SEX       CHAR(1)                ,
       BIRTHDATE DATE                   ,
       SALARY    DECIMAL(9, 2)          ,
       BONUS     DECIMAL(9, 2)          ,
       COMM      DECIMAL(9, 2)          );
```

This statement also creates a referential constraint between the foreign key in YEMP (WORKDEPT) and the primary key in YDEPT (DEPTNO). It also restricts all phone numbers to unique numbers.

If you want to change a table definition after you create it, use the ALTER TABLE statement with a RENAME clause. If you want to change a table name after you create it, use the RENAME statement.

You can change a table definition by using the ALTER TABLE statement only in certain ways. For example, you can add and drop constraints on columns in a table. You can also change the data type of a column within character data types, within numeric data types, and within graphic data types. You can add a column to a table. However, you cannot use the ALTER TABLE statement to drop a column from a table.

Related tasks

Altering Db2 tables (Db2 Administration Guide)

Related reference

ALTER TABLE (Db2 SQL)

RENAME (Db2 SQL)

Creating created temporary tables

Use created temporary tables when you need to store data for only the life of an application process, but you want to share the table definition.

About this task

Db2 does not perform logging and locking operations for created temporary tables. Therefore, SQL statements that use these tables can execute queries efficiently.

Each application process has its own instance of the created temporary table.

An instance of a created temporary table exists at the current server until one of the following actions occurs:

- The application process ends.
- The remote server connection through which the instance was created terminates.
- The unit of work in which the instance was created completes.

  When you run a ROLLBACK statement, Db2 deletes the instance of the created temporary table. When you run a COMMIT statement, Db2 deletes the instance of the created temporary table unless a cursor for accessing the created temporary table is defined with the WITH HOLD clause and is open.

You create the definition of a created temporary table using the SQL CREATE GLOBAL TEMPORARY TABLE statement.

Procedure

To create a created temporary table:

1. Define the table by issuing CREATE GLOBAL TEMPORARY TABLE statement.
   For example, the following statement creates the definition of a table called TEMPPROD:

   ```
   CREATE GLOBAL TEMPORARY TABLE TEMPPROD
     (SERIAL       CHAR(8)     NOT NULL,
      DESCRIPTION  VARCHAR(60) NOT NULL,
      MFGCOST      DECIMAL(8,2),
      MFGDEPT      CHAR(3),
      MARKUP       SMALLINT,
      SALESDEPT    CHAR(3),
      CURDATE      DATE        NOT NULL);
   ```

   You can also create this same definition by copying the definition of a base table (named PROD) by using the LIKE clause:

   ```
   CREATE GLOBAL TEMPORARY TABLE TEMPPROD LIKE PROD;
   ```

   Restriction: You cannot use the MERGE statement with created temporary tables.

   The SQL statements in the examples create identical definitions for the TEMPPROD table, but these tables differ slightly from the PROD sample table PROD. The PROD sample table contains two columns, DESCRIPTION and CURDATE, that are defined as NOT NULL WITH DEFAULT. Because created temporary tables do not support non-null default values, the DESCRIPTION and CURDATE columns in the TEMPPROD table are defined as NOT NULL and do not have defaults.

   After you run one of the two CREATE statements, the definition of TEMPPROD exists, but no instances of the table exist.
2. Create an instance of the created temporary table by using it in an application.

   Db2 creates an instance of the table when it is specified in one of the following SQL statements:

   - OPEN
   - SELECT
   - INSERT
   - DELETE

   For example, suppose that you defined TEMPROD as described the previous step and then run an application that contains the following statements:

   ```
   EXEC SQL DECLARE C1 CURSOR FOR SELECT * FROM TEMPPROD;
   EXEC SQL INSERT INTO TEMPPROD SELECT * FROM PROD;
   EXEC SQL OPEN C1;
   ⋮
   EXEC SQL COMMIT;
   ⋮
   EXEC SQL CLOSE C1;
   ```

   When you run the INSERT statement, Db2 creates an instance of TEMPPROD and populates that instance with rows from table PROD. When the COMMIT statement runs, Db2 deletes all rows from TEMPPROD. However, assume that you change the declaration of cursor C1 to the following declaration:

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD
  FOR SELECT * FROM TEMPPROD;
```

In this case, Db2 does not delete the contents of TEMPPROD until the application ends because C1, a cursor that is defined with the WITH HOLD clause, is open when the COMMIT statement runs. In either case, Db2 drops the instance of TEMPPROD when the application ends.

3. When the table is no longer needed, issue a DROP statement .
   For example, to drop the definition of TEMPPROD, you must run the following statement:

```
DROP TABLE TEMPPROD;
```

Related reference

CREATE GLOBAL TEMPORARY TABLE (Db2 SQL)

DROP (Db2 SQL)

Temporary tables

Use temporary tables when you need to store data for only the duration of an application process. Depending on whether you want to share the table definition, you can create a created temporary table or a declared temporary table.

The two kinds of temporary tables are:

- Created temporary tables, which you define using a CREATE GLOBAL TEMPORARY TABLE statement
- Declared temporary tables, which you define using a DECLARE GLOBAL TEMPORARY TABLE statement

SQL statements that use temporary tables can run faster because of the following reasons:

- For created temporary tables, Db2 provides no logging. For declared temporary tables, Db2 provides limited logging that can be further limited by the NOT LOGGED option of the DECLARE GLOBAL TEMPORARY TABLE statement.
- For created temporary tables, Db2 provides no locking. For declared temporary tables, Db2 provides limited locking.

Temporary tables are especially useful when you need to sort or query intermediate result tables that contain a large number of rows, but you want to store only a small subset of those rows permanently.

Temporary tables can also return result sets from stored procedures. The following topics provide more details about created temporary tables and declared temporary tables:

- “Creating created temporary tables” on page 137
- “Creating declared temporary tables” on page 139

For more information, see “Writing an external procedure to return result sets to a distributed client” on page 279.

Creating declared temporary tables

Use declared temporary tables when you need to store data for only the life of an application process and do not need to share the table definition. The definition of this table exists only while the application process runs. Db2 performs limited logging and locking operations for declared temporary tables.

Before you begin

Before you can define declared temporary tables, you must have a WORKFILE database that has at least one table space with a 32-KB page size.

About this task

You create an instance of a declared temporary table by using the SQL DECLARE GLOBAL TEMPORARY TABLE statement. That instance is known only to the application process in which the table is declared, so you can declare temporary tables with the same name in different applications. The qualifier for a declared temporary table is SESSION.

To create a declared temporary table, specify the DECLARE GLOBAL TEMPORARY TABLE statement. In that statement, specify the columns that the table is to contain by performing one of the following actions:

Procedure

To create a declared temporary table:

1. Issue a DECLARE GLOBAL TEMPORARY TABLE statement.

   In that statement, you an specify the columns that the table is to contain by performing one of the following actions:

   - Specify all the columns in the table. For example, the following statement defines a declared temporary table called TEMPPROD by explicitly specifying the columns.

     ```
     DECLARE GLOBAL TEMPORARY TABLE TEMPPROD
       (SERIAL       CHAR(8)     NOT NULL WITH DEFAULT '99999999',
        DESCRIPTION  VARCHAR(60) NOT NULL,
        PRODCOUNT    INTEGER GENERATED ALWAYS AS IDENTITY,
        MFGCOST      DECIMAL(8,2),
        MFGDEPT      CHAR(3),
        MARKUP       SMALLINT,
        SALESDEPT    CHAR(3),
        CURDATE      DATE        NOT NULL);
     ```

     Unlike columns of created temporary tables, columns of declared temporary tables can include the WITH DEFAULT clause.

   - Use a LIKE clause to copy the definition of a base table, created temporary table, or view. For example, the following statement defines a declared temporary table called TEMPPROD by copying the definition of a base table. The base table has an identity column that the declared temporary table also uses as an identity column.

     ```
     DECLARE GLOBAL TEMPORARY TABLE TEMPPROD LIKE BASEPROD
       INCLUDING IDENTITY COLUMN ATTRIBUTES;
     ```

   - If the base table, created temporary table, or view from which you select columns has identity columns, you can specify that the corresponding columns in the declared temporary table are also identity columns. To include these identity columns, specify the INCLUDING IDENTITY COLUMN ATTRIBUTES clause when you define the declared temporary table.

     If the source table has a row change timestamp column, you can specify that those column attributes are inherited in the declared temporary table by specifying INCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES.

   - Use a fullselect to choose specific columns from a base table, created temporary table, or view. For example, the following statement defines a declared temporary table called TEMPPROD by selecting columns from a view. The view has an identity column that the declared temporary table also uses as an identity column. The declared temporary table inherits its default column values from the default column values of a base table on which the view is based.

     ```
     DECLARE GLOBAL TEMPORARY TABLE TEMPPROD
       AS (SELECT * FROM PRODVIEW)
       DEFINITION ONLY
       INCLUDING IDENTITY COLUMN ATTRIBUTES
       INCLUDING COLUMN DEFAULTS;
     ```

     If you want the declared temporary table columns to inherit the defaults for columns of the table or view that is named in the fullselect, specify the INCLUDING COLUMN DEFAULTS clause. If you want the declared temporary table columns to have default values that correspond to their data types, specify the USING TYPE DEFAULTS clause.

Db2 creates an empty instance of a declared temporary table.

2. Complete one of the following actions:
   - Populate the declared temporary table by using INSERT statements
   - Modify the table using searched or positioned UPDATE or DELETE statements
   - Query the table using SELECT statements
   - Create indexes on the declared temporary table
3. After you run a DECLARE GLOBAL TEMPORARY TABLE statement, the definition of the declared temporary table exists as long as the application process runs. If you need to delete the definition before the application process completes, issue a DROP TABLE statement.
   For example, to drop the definition of TEMPPROD, run the following statement:

```
DROP TABLE SESSION.TEMPPROD;
```

Example

The ON COMMIT clause that you specify in the DECLARE GLOBAL TEMPORARY TABLE statement determines whether Db2 keeps or deletes all the rows from the table when you run a COMMIT statement in an application with a declared temporary table. ON COMMIT DELETE ROWS, which is the default, causes all rows to be deleted from the table at a commit point, unless a held cursor is open on the table at the commit point. ON COMMIT PRESERVE ROWS causes the rows to remain past the commit point.

For example Suppose that you run the following statement in an application program:

```
EXEC SQL DECLARE GLOBAL TEMPORARY TABLE TEMPPROD
  AS (SELECT * FROM BASEPROD)
  DEFINITION ONLY
  INCLUDING IDENTITY COLUMN ATTRIBUTES
  INCLUDING COLUMN DEFAULTS
  ON COMMIT PRESERVE ROWS;
EXEC SQL INSERT INTO SESSION.TEMPPROD SELECT * FROM BASEPROD;
⋮
EXEC SQL COMMIT;
⋮
```

When Db2 runs the preceding DECLARE GLOBAL TEMPORARY TABLE statement, Db2 creates an empty instance of TEMPPROD. The INSERT statement populates that instance with rows from table BASEPROD. The qualifier, SESSION, must be specified in any statement that references TEMPPROD. When Db2 executes the COMMIT statement, Db2 keeps all rows in TEMPPROD because TEMPPROD is defined with ON COMMIT PRESERVE ROWS. When the program ends, Db2 drops TEMPPROD.

Related reference

DECLARE GLOBAL TEMPORARY TABLE (Db2 SQL)

Providing a unique key for a table

Use ROWID columns or identity columns to store unique values for each row in a table.

About this task

Question: How can I provide a unique identifier for a table that has no unique column?

Answer: Add a column with the data type ROWID or an identity column. ROWID columns and identity columns contain a unique value for each row in the table. You can define the column as GENERATED ALWAYS, which means that you cannot insert values into the column, or GENERATED BY DEFAULT, which means that Db2 generates a value if you do not specify one. If you define the ROWID or identity column as GENERATED BY DEFAULT, you need to define a unique index that includes only that column to guarantee uniqueness.

Fixing tables with incomplete definitions

If a table has an incomplete definition, you cannot load the table, insert data, retrieve data, update data, or delete data. You can however drop the table, create the primary index, and drop or create other indexes.

Before you begin

To check if a table has an incomplete definition, look at the STATUS column in SYSIBM.SYSTABLES. The value I indicates that the definition is incomplete.

About this task

A table definition is incomplete in any of the following circumstances:

- **If the table is defined with a primary or unique key** and all of the following conditions are true:
  - The table space for the table was explicitly created.
  - The statement is not being run with schema processor.
  - The table does not have a primary or unique index for the defined primary or unique key.
- **If the table has a ROWID column that is defined as generated by default** and all of the following conditions are true:
  - The table space for the table was explicitly created.
  - The SET CURRENT RULES special register is not set to STD.
  - No unique index is defined on the ROWID column.
- **If the table has a LOB column** and all of the following conditions are true:
  - The table space for the table was explicitly created.
  - The SET CURRENT RULES special register is not set to STD.
  - No all auxiliary LOB objects are defined for the LOB column.

Procedure

To complete the definition of a table, use one of the following actions:

- Create a primary index or alter the table to drop the primary key.
- Create a unique index on the unique key or alter the table to drop the unique key.
- Defining a unique index on the ROWID column.
- Create the necessary LOB objects.

Example

To create the primary index for the project activity table, issue the following SQL statement:

```
CREATE UNIQUE INDEX XPROJAC1
  ON DSN8B10.PROJACT (PROJNO, ACTNO, ACSTDATE);
```

RENAME TABLE in a table maintenance scenario

The RENAME TABLE statement is useful when you need to temporarily take a table offline for maintenance that involves structural changes to the table. Applications can continue to run against another copy of the table until maintenance is complete.

One way of accomplishing this is refer to the name of the table as an unqualified name in all applications. The unqualified table name is implicitly qualified by the content of the CURRENT SCHEMA special register. You set CURRENT SCHEMA to the schema of the real table to cause applications to access the real table. Before you take the real table offline, you change the CURRENT SCHEMA special register to the name of the schema for the alternate copy of the table. When all applications are running with the alternate copy of the table, the real table can be modified. An example of such a modification is adding a column to the table.

Later, when table maintenance is complete, you can set the CURRENT SCHEMA special register to the name of the schema for the real table to cause all applications to switch back to using the real table.

Related reference

RENAME (Db2 SQL)

CURRENT SCHEMA (Db2 SQL)

Dropping tables

When you drop a table, you delete the data and the table definition. You also delete all synonyms, views, indexes, referential constraints, and check constraints that are associated with that table.

About this task

The following SQL statement drops the YEMP table:

```
DROP TABLE YEMP;
```

Use the DROP TABLE statement with care: Dropping a table is **not** equivalent to deleting all its rows. When you drop a table, you lose more than its data and its definition. You lose all synonyms, views, indexes, and referential and check constraints that are associated with that table. You also lose all authorities that are granted on the table.

Related reference

DROP (Db2 SQL)

Defining a view

A *view* is a named specification of a result table. Use views to control which users have access to certain data or to simplify writing SQL statements.

About this task

Use the CREATE VIEW statement to define a view and give the view a name, just as you do for a table. The view that is created with the following statement shows each department manager's name with the department data in the DSN8B10.DEPT table.

```
CREATE VIEW VDEPTM AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
    FROM DSN8B10.DEPT, DSN8B10.EMP
      WHERE DSN8B10.EMP.EMPNO = DSN8B10.DEPT.MGRNO;
```

When a program accesses the data that is defined by a view, Db2 uses the view definition to return a set of rows that the program can access with SQL statements.

To see the departments that are administered by department D01 and the managers of those departments, run the following statement, which returns information from the VDEPTM view:

```
SELECT DEPTNO, LASTNAME
  FROM VDEPTM
    WHERE ADMRDEPT = 'D01';
```

When you create a view, you can reference the SESSION_USER and CURRENT SQLID special registers in the CREATE VIEW statement. When referencing the view, Db2 uses the value of the SESSION_USER or CURRENT SQLID special register that belongs to the user of the SQL statement (SELECT, UPDATE, INSERT, or DELETE) rather than the creator of the view. In other words, a reference to a special register in a view definition refers to its run time value.

You can specify a period specification for a view, subject to certain restrictions. Also, for a view that references an application-period temporal table or a bitemporal table, you can specify a period clause for an update or delete operation on the view.

A column in a view might be based on a column in a base table that is an identity column. The column in the view is also an identity column,**except** under any of the following circumstances:

- The column appears more than once in the view.
- The view is based on a join of two or more tables.
- The view is based on the union of two or more tables.
- Any column in the view is derived from an expression that refers to an identity column.

You can use views to limit access to certain kinds of data, such as salary information. Alternatively, you can use the IMPLICITLY HIDDEN clause of a CREATE TABLE statement define a column of a table to be hidden from some operations.

You can also use views for the following actions:

- Make a subset of a table's data available to an application. For example, a view based on the employee table might contain rows only for a particular department.
- Combine columns from two or more tables and make the combined data available to an application. By using a SELECT statement that matches values in one table with those in another table, you can create a view that presents data from both tables. However, you can **only select** data from this type of view. **You cannot update, delete, or insert data using a view that joins two or more tables.**
- Combine rows from two or more tables and make the combined data available to an application. By using two or more subselects that are connected by a set operator such as UNION, you can create a view that presents data from several tables. However, you can **only select** data from this type of view. **You cannot update, delete, or insert data using a view that contains UNION operations.**
- Present computed data, and make the resulting data available to an application. You can compute such data using any function or operation that you can use in a SELECT statement.

Related tasks

Changing data by using views that reference temporal tables (Db2 Administration Guide)

Related reference

CREATE VIEW (Db2 SQL)

Related information

Implementing Db2 views (Db2 Administration Guide)

Views

A view does not contain data; it is a stored definition of a set of rows and columns. A view can present any or all of the data in one or more tables.

Although you cannot modify an existing view, you can drop it and create a new one if your base tables change in a way that affects the view. Dropping and creating views does not affect the base tables or their data.

Restrictions when changing data through a view

Some views are read-only and thus cannot be used to update the table data. For those views that are updatable, several restrictions apply.

Consider the following restrictions when changing data through a view:

- You must have the appropriate authorization to insert, update, or delete rows using the view.
- When you use a view to insert a row into a table, the view definition must specify all the columns in the base table that do not have a default value. The row that is being inserted must contain a value for each of those columns.

- Views that you can use to update data are subject to the same referential constraints and check constraints as the tables that you used to define the views.

  You can use the WITH CHECK option of the CREATE VIEW statement to specify the constraint that every row that is inserted or updated through the view must conform to the definition of the view. You can select every row that is inserted or updated through a view that is created with the WITH CHECK option.
- For an update operation on a view that references an application-period temporal table or a bitemporal table, the result table of the outer fullselect of the view definition, explicitly or implicitly, must include the start and end columns of the BUSINESS_TIME period.
- For an update or delete operation on a view that references an application-period temporal table or a bitemporal table, the view must not be defined with an INSTEAD OF trigger.

For complex views, you can make insert, update and delete operations possible by defining INSTEAD OF triggers.

Related tasks

Inserting, updating, and deleting data in views by using INSTEAD OF triggers
INSTEAD OF triggers are triggers that execute instead of the INSERT, UPDATE, or DELETE statement that activates the trigger. You can define these triggers on views only. Use INSTEAD OF triggers to insert, update, and delete data in complex views.

Changing data by using views that reference temporal tables (Db2 Administration Guide)

Related reference

CREATE VIEW (Db2 SQL)

Dropping a view

When you drop a view, you also drop all views that are defined on that view. The base table is not affected.

Example

The following SQL statement drops the VDEPTM view:

```
DROP VIEW VDEPTM;
```

Creating a common table expression

Creating a common table expression saves you the overhead of creating and dropping a regular view that you need to use only once. Also, during statement preparation, Db2 does not need to access the catalog for the view, which saves you additional overhead.

About this task

Use the WITH clause to create a common table expression.

Procedure

To created a common table expression use one of the following approaches:

- Specify a WITH clause at the beginning of a SELECT statement.
  For example, the following statement finds the department with the highest total pay. The query involves two levels of aggregation. First, you need to determine the total pay for each department by using the SUM function and order the results by using the GROUP BY clause. You then need to find the department with highest total pay based on the total pay for each department.

```
WITH DTOTAL (workdept, totalpay) AS
        (SELECT deptno, sum(salary+bonus)
             FROM  DSN8810.EMP
             GROUP BY workdept)
     SELECT workdept
       FROM DTOTAL
```

```
       WHERE totalpay = (SELECT max(totalpay)
                               FROM  DTOTAL);
```

The result table for the common table expression, DTOTAL, contains the department number and total pay for each department in the employee table. The fullselect in the previous example uses the result table for DTOTAL to find the department with the highest total pay. The result table for the entire statement looks similar to the following results:

```
WORKDEPT
======
D11
```

- Use common table expressions by specifying WITH before a fullselect in a CREATE VIEW statement.

  This technique is useful if you need to use the results of a common table expression in more than one query.

  For example, the following statement finds the departments that have a greater-than-average total pay and saves the results as the view RICH_DEPT:

```
CREATE VIEW RICH_DEPT (workdept) AS
         WITH DTOTAL (workdept, totalpay) AS
          (SELECT workdept, sum(salary+bonus)
              FROM DSN8B10.EMP
              GROUP BY workdept)
         SELECT workdept
           FROM DTOTAL
            WHERE totalpay > (SELECT AVG(totalpay)
                                    FROM DTOTAL);
```

  The fullselect in the previous example uses the result table for DTOTAL to find the departments that have a greater-than-average total pay. The result table is saved as the RICH_DEPT view and looks similar to the following results:

```
WORKDEPT
======
A00
D11
D21
```

- Use common table expressions by specifying WITH before a fullselect in an INSERT statement. For example, the following statement uses the result table for VITALDEPT to find the manager's number for each department that has a greater-than-average number of senior engineers. Each manager's number is then inserted into the vital_mgr table.

```
INSERT INTO vital_mgr (mgrno)
      WITH VITALDEPT (workdept, se_count)  AS
          (SELECT workdept, count(*)
              FROM DSN8B10.EMP
              WHERE job = 'senior engineer'
               GROUP BY workdept)
      SELECT d.manager
         FROM DSN8B10.DEPT d, VITALDEPT s
         WHERE d.workdept = s.workdept
             AND s.se_count  >  (SELECT  AVG(se_count)
                                     FROM VITALDEPT);
```

Related reference

common-table-expression (Db2 SQL)

Common table expressions

A *common table expression* is like a temporary view that is defined and used for the duration of an SQL statement.

You can define a common table expression wherever you can have a fullselect statement. For example, you can include a common table expression in a SELECT, INSERT, SELECT INTO, or CREATE VIEW statement.

Each common table expression must have a unique name and be defined only once. However, you can reference a common table expression many times in the same SQL statement. Unlike regular views or nested table expressions, which derive their result tables for each reference, all references to common table expressions in a given statement share the same result table.

You can use a common table expression in the following situations:

- When you want to avoid creating a view (when general use of the view is not required, and positioned updates or deletes are not used)
- When the result table is based on host variables
- When the same result table needs to be shared in a fullselect
- When the results need to be derived using recursion

Related reference

common-table-expression (Db2 SQL)

Examples of recursive common table expressions

Recursive SQL is very useful in bill of materials (BOM) applications.

Consider a table of parts with associated subparts and the quantity of subparts required by each part. For more information about recursive SQL, refer to "Creating recursive SQL by using common table expressions" on page 387.

For the examples in this topic, create the following table:

```
CREATE TABLE PARTLIST
     (PART VARCHAR(8),
      SUBPART VARCHAR(8),
      QUANTITY INTEGER);
```

Assume that the PARTLIST table is populated with the values that are in the following table:

Table 38. PARTLIST table

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10

PART	SUBPART	QUANTITY
07	14	8
07	12	8

Table 38. PARTLIST table (continued)

Example 1: Single level explosion:

Single level explosion answers the question, "What parts are needed to build the part identified by '01'?". The list will include the direct subparts, subparts of the subparts and so on. However, if a part is used multiple times, its subparts are only listed once.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
     (SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
        FROM PARTLIST ROOT
        WHERE ROOT.PART = '01'
     UNION ALL
        SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
           FROM RPL PARENT, PARTLIST CHILD
           WHERE PARENT.SUBPART = CHILD.PART)
SELECT DISTINCT PART, SUBPART, QUANTITY
    FROM RPL
    ORDER BY PART, SUBPART, QUANTITY;
```

The preceding query includes a common table expression, identified by the name RPL, that expresses the recursive part of this query. It illustrates the basic elements of a recursive common table expression.

The first operand (fullselect) of the UNION, referred to as the initialization fullselect, gets the direct subparts of part '01'. The FROM clause of this fullselect refers to the source table and will never refer to itself (RPL in this case). The result of this first fullselect goes into the common table expression RPL. As in this example, the UNION must always be a UNION ALL.

The second operand (fullselect) of the UNION uses RPL to compute subparts of subparts by using the FROM clause to refer to the common table expression RPL and the source table PARTLIST with a join of a part from the source table (child) to a subpart of the current result contained in RPL (parent). The result goes then back to RPL again. The second operand of UNION is used repeatedly until no more subparts exist.

The SELECT DISTINCT in the main fullselect of this query ensures the same part/subpart is not listed more than once.

The result of the query is shown in the following table:

Table 39. Result table for example 1

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10

Table 39. Result table for example 1 (continued)

PART	SUBPART	QUANTITY
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

Observe in the result that part '01' contains subpart '02' which contains subpart '06' and so on. Further, notice that part '06' is reached twice, once through part '01' directly and another time through part '02'. In the output, however, the subparts of part '06' are listed only once (this is the result of using a SELECT DISTINCT).

Remember that with recursive common table expressions it is possible to introduce an infinite loop. In this example, an infinite loop would be created if the search condition of the second operand that joins the parent and child tables was coded as follows:

```
WHERE PARENT.SUBPART = CHILD.SUBPART
```

This infinite loop is created by not coding what is intended. You should carefully determining what to code so that there is a definite end of the recursion cycle.

The result produced by this example could be produced in an application program without using a recursive common table expression. However, such an application would require coding a different query for every level of recursion. Furthermore, the application would need to put all of the results back in the database to order the final result. This approach complicates the application logic and does not perform well. The application logic becomes more difficult and inefficient for other bill of material queries, such as summarized and indented explosion queries.

Example 2: Summarized explosion:

A summarized explosion answers the question, "What is the total quantity of each part required to build part '01'?" The main difference from a single level explosion is the need to aggregate the quantities. A single level explosion indicates the quantity of subparts required for the part whenever it is required. It does not indicate how many of each subpart is needed to build part '01'.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
           (
              SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
                 FROM PARTLIST ROOT
                 WHERE ROOT.PART = '01'
             UNION ALL
              SELECT PARENT.PART, CHILD.SUBPART,
                     PARENT.QUANTITY*CHILD.QUANTITY
                 FROM RPL PARENT, PARTLIST CHILD
                 WHERE PARENT.SUBPART = CHILD.PART
           )
SELECT PART, SUBPART, SUM(QUANTITY) AS "Total QTY Used"
   FROM RPL
   GROUP BY PART, SUBPART
   ORDER BY PART, SUBPART;
```

In the preceding query, the select list of the second operand of the UNION in the recursive common table expression, identified by the name RPL, shows the aggregation of the quantity. To determine how many of each subpart is used, the quantity of the parent is multiplied by the quantity per parent of a child. If a part is used multiple times in different places, it requires another final aggregation. This is done by the grouping the parts and subparts in the common table expression RPL and using the SUM column function in the select list of the main fullselect.

The result of the query is shown in the following table:

Table 40. Result table for example 2

PART	SUBPART	Total QTY Used
01	02	2
01	03	3
01	04	4
01	05	14
01	06	15
01	07	18
01	08	40
01	09	44
01	10	140
01	11	140
01	12	294
01	13	150
01	14	144

Consider the total quantity for subpart '06'. The value of 15 is derived from a quantity of 3 directly for part '01' and a quantity of 6 for part '02' which is needed two times by part '01'.

Example 3: Controlling depth:

You can control the depth of a recursive query to answer the question, "What are the first two levels of parts that are needed to build part '01'?" For the sake of clarity in this example, the level of each part is included in the result table.

```
WITH RPL (LEVEL, PART, SUBPART, QUANTITY) AS
       (
         SELECT 1, ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
           FROM PARTLIST ROOT
           WHERE ROOT.PART = '01'
     UNION ALL
         SELECT PARENT.LEVEL+1, CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
           FROM RPL PARENT, PARTLIST CHILD
           WHERE PARENT.SUBPART = CHILD.PART
             AND PARENT.LEVEL < 2
        )
SELECT PART, LEVEL, SUBPART, QUANTITY
  FROM RPL;
```

This query is similar to the query in example 1. The column LEVEL is introduced to count the level each subpart is from the original part. In the initialization fullselect, the value for the LEVEL column is initialized to 1. In the subsequent fullselect, the level from the parent table increments by 1. To control the number of levels in the result, the second fullselect includes the condition that the level of the parent must be less than 2. This ensures that the second fullselect only processes children to the second level.

The result of the query is shown in the following table:

Table 41. Result table for example 3

PART	LEVEL	SUBPART	QUANTITY
01	1	02	2
01	1	03	3

Table 41. Result table for example 3 (continued)

PART	LEVEL	SUBPART	QUANTITY
01	1	04	4
01	1	06	3
02	2	05	7
02	2	06	6
03	2	07	6
04	2	08	10
04	2	09	11
06	2	12	10
06	2	13	10

Creating a trigger

A *trigger* is a set of SQL statements that execute when a certain event occurs in a table or view. Use triggers to control changes in Db2 databases. Triggers are more powerful than constraints because they can monitor a broader range of changes and perform a broader range of actions.

About this task

Using triggers for active data:

For example, a constraint can disallow an update to the salary column of the employee table if the new value is over a certain amount. A trigger can monitor the amount by which the salary changes, as well as the salary value. If the change is above a certain amount, the trigger might substitute a valid value and call a user-defined function to send a notice to an administrator about the invalid update.

Triggers also move application logic into Db2, which can result in faster application development and easier maintenance. For example, you can write applications to control salary changes in the employee table, but each application program that changes the salary column must include logic to check those changes. A better method is to define a trigger that controls changes to the salary column. Then Db2 does the checking for any application that modifies salaries.

Example of creating and using a trigger:

Triggers automatically execute a set of SQL statements whenever a specified event occurs. These SQL statements can perform tasks such as validation and editing of table changes, reading and modifying tables, or invoking functions or stored procedures that perform operations both inside and outside Db2.

You create triggers using the CREATE TRIGGER statement. The following figure shows an example of a CREATE TRIGGER statement.

```
                1
CREATE TRIGGER REORDER
   2      3                                4
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
                      5
  REFERENCING NEW AS N_ROW
   6
  FOR EACH ROW MODE DB2SQL
   7
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
   8
  BEGIN ATOMIC
  CALL ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO);
  END
```

The parts of this trigger are:

1
Trigger name (REORDER)

2
Trigger activation time (AFTER)

3
Triggering event (UPDATE)

4
Subject table name (PARTS)

5
New transition variable correlation name (N_ROW)

6
Granularity (FOR EACH ROW)

7
Trigger condition (WHEN...)

8
Trigger body (BEGIN ATOMIC...END;)

When you execute this CREATE TRIGGER statement, Db2 creates a trigger package called REORDER and associates the trigger package with table PARTS. Db2 records the timestamp when it creates the trigger. If you define other triggers on the PARTS table, Db2 uses this timestamp to determine which trigger to activate first when the triggering event occurs. The trigger is now ready to use.

After Db2 updates columns ON_HAND or MAX_STOCKED in any row of table PARTS, trigger REORDER is activated. The trigger calls a stored procedure called ISSUE_SHIP_REQUEST if, after a row is updated, the quantity of parts on hand is less than 10% of the maximum quantity stocked. In the trigger condition, the qualifier N_ROW represents a value in a modified row after the triggering event.

When you no longer want to use trigger REORDER, you can delete the trigger by executing the statement:

```
DROP TRIGGER REORDER;
```

Executing this statement drops trigger REORDER and its associated trigger package named REORDER.

If you drop table PARTS, Db2 also drops trigger REORDER and its trigger package.

Parts of a trigger:

A trigger contains the following parts:

- trigger name
- subject table
- trigger activation time
- triggering event
- granularity
- correlation names for transition variables and transition tables
- triggered action that consists of an optional search condition and a trigger body

Trigger name:

Use an ordinary identifier to name your trigger. You can use a qualifier or let Db2 determine the qualifier. When Db2 creates a trigger package for the trigger, it uses the qualifier for the collection ID of the trigger package. Db2 uses these rules to determine the qualifier:

- If you use static SQL to execute the CREATE TRIGGER statement, Db2 uses the authorization ID in the bind option QUALIFIER for the package that contains the CREATE TRIGGER statement. If the bind command does not include the QUALIFIER option, Db2 uses the owner of the package.

- If you use dynamic SQL to execute the CREATE TRIGGER statement, Db2 uses the authorization ID in special register CURRENT SCHEMA.

Subject table or view:

When you perform an insert, update, or delete operation on this table or view, the trigger is activated. You must name a local table or view in the CREATE TRIGGER statement. You cannot define a trigger on a catalog table.

Trigger activation time:

The two choices for trigger activation time are NO CASCADE BEFORE and AFTER. NO CASCADE BEFORE means that the trigger is activated before Db2 makes any changes to the subject table, and that the triggered action does not activate any other triggers. AFTER means that the trigger is activated after Db2 makes changes to the subject table and can activate other triggers. Triggers with an activation time of NO CASCADE BEFORE are known as before triggers. Triggers with an activation time of AFTER are known as after triggers.

Triggering event:

Every trigger is associated with an event. A trigger is activated when the triggering event occurs in the subject table or view. The triggering event is one of the following SQL operations:

- insert
- update
- delete

A triggering event can also be an update or delete operation that occurs as the result of a referential constraint with ON DELETE SET NULL or ON DELETE CASCADE.

A trigger can be activated by a MERGE statement for insert and update operations.

Triggers are not activated as the result of updates made to tables by Db2 utilities, with the exception of the LOAD utility when it is specified with the RESUME YES and SHRLEVEL CHANGE options.

When the triggering event for a trigger is an update operation, the trigger is called an update trigger. Similarly, triggers for insert operations are called insert triggers, and triggers for delete operations are called delete triggers.

The SQL statement that performs the triggering SQL operation is called the triggering SQL statement. Each triggering event is associated with one subject table or view and one SQL operation.

The following trigger is defined with an insert triggering event:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

If the triggering SQL operation is an update operation, the event can be associated with specific columns of the subject table. In this case, the trigger is activated only if the update operation updates any of the specified columns.

The following trigger, PAYROLL1, which invokes user-defined function named PAYROLL_LOG, is activated only if an update operation is performed on the SALARY or BONUS column of table PAYROLL:

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE OF SALARY, BONUS ON PAYROLL
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE', CURRENT TIME, CURRENT DATE));
  END
```

Granularity:

The triggering SQL statement might modify multiple rows in the table. The granularity of the trigger determines whether the trigger is activated only once for the triggering SQL statement or once for every row that the SQL statement modifies. The granularity values are:

- FOR EACH ROW

  The trigger is activated once for each row that Db2 modifies in the subject table or view. If the triggering SQL statement modifies no rows, the trigger is not activated. However, if the triggering SQL statement updates a value in a row to the same value, the trigger is activated. For example, if an UPDATE trigger is defined on table COMPANY_STATS, the following SQL statement will activate the trigger.

  ```
  UPDATE COMPANY_STATS SET NBEMP = NBEMP;
  ```

- FOR EACH STATEMENT

  The trigger is activated once when the triggering SQL statement executes. The trigger is activated even if the triggering SQL statement modifies no rows.

Triggers with a granularity of FOR EACH ROW are known as row triggers. Triggers with a granularity of FOR EACH STATEMENT are known as statement triggers. Statement triggers can only be after triggers.

The following statement is an example of a row trigger:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

Trigger NEW_HIRE is activated once for every row inserted into the employee table.

Transition variables:

When you code a row trigger, you might need to refer to the values of columns in each updated row of the subject table or view. To do this, specify a correlation name (to use when referencing transition variables) in the REFERENCING clause of your CREATE TRIGGER statement. The two types of transition variables are:

- Old transition variables capture the values of columns before the triggering SQL statement updates them. You can use the REFERENCING OLD clause to define a correlation name for referencing old transition variables for update and delete triggers.
- New transition variables capture the values of columns after the triggering SQL statement updates them. You can use the REFERENCING NEW clause to define a correlation name for referencing new transition variables for update and insert triggers.

Transition variables can be referenced anywhere in an SQL statement where an expression or variable can be specified in triggers.

The following example uses transition variables and invocations of the IDENTITY_VAL_LOCAL function to access values that are assigned to identity columns.

Suppose that you have created tables T and S, with the following definitions:

```
CREATE TABLE T
  (ID SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 100),
   C2 SMALLINT,
   C3 SMALLINT,
   C4 SMALLINT);
```

```
CREATE TABLE S
  (ID SMALLINT GENERATED ALWAYS AS IDENTITY,
   C1 SMALLINT);
```

Define a before insert trigger on T that uses the IDENTITY_VAL_LOCAL built-in function to retrieve the current value of identity column ID, and uses transition variables to update the other columns of T with the identity column value.

```
CREATE TRIGGER TR1
  NO CASCADE BEFORE INSERT
  ON T REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET N.C3 =N.ID;
    SET N.C4 =IDENTITY_VAL_LOCAL();
    SET N.ID =N.C2 *10;
    SET N.C2 =IDENTITY_VAL_LOCAL();
  END
```

Now suppose that you execute the following INSERT statement:

```
INSERT INTO S (C1) VALUES (5);
```

This statement inserts a row into S with a value of 5 for column C1 and a value of 1 for identity column ID. Next, suppose that you execute the following SQL statement, which activates trigger TR1:

```
INSERT INTO T (C2)
  VALUES (IDENTITY_VAL_LOCAL());
```

This insert statement, and the subsequent activation of trigger TR1, have the following results:

- The INSERT statement obtains the most recent value that was assigned to an identity column (1), and inserts that value into column C2 of table T. 1 is the value that Db2 inserted into identity column ID of table S.
- When the INSERT statement executes, Db2 inserts the value 100 into identity column ID column of C2.
- The first statement in the body of trigger TR1 inserts the value of transition variable N.ID (100) into column C3. N.ID is the value that identity column ID contains *after* the INSERT statement executes.
- The second statement in the body of trigger TR1 inserts the null value into column C4. By definition, the result of the IDENTITY_VAL_LOCAL function in the triggered action of a before insert trigger is the null value.
- The third statement in the body of trigger TR1 inserts 10 times the value of transition variable N.C2 (10*1) into identity column ID of table T. N.C2 is the value that column C2 contains *after* the INSERT is executed.
- The fourth statement in the body of trigger TR1 inserts the null value into column C2. By definition, the result of the IDENTITY_VAL_LOCAL function in the triggered action of a before insert trigger is the null value.

Transition tables:

If you want to refer to the entire set of rows that a triggering SQL statement modifies, rather than to individual rows, use a transition table. Like transition variables, a correlation name (to refer to the columns of the transition table) can appear in the REFERENCING clause of a CREATE TRIGGER statement. The names for those columns are the same as the name of the column in the table or view that the trigger is defined for. Transition tables are valid for both row triggers and statement triggers. The two types of transition tables are:

- Old transition tables, specified with the OLD TABLE *transition-table-name* clause, capture the values of columns before the triggering SQL statement updates them. You can define old transition tables for update and delete triggers.
- New transition tables, specified with the NEW TABLE *transition-table-name* clause, capture the values of columns after the triggering SQL statement updates them. You can define new transition variables for update and insert triggers.

The scope of old and new transition table names is the trigger body. If correlation names are specified for both old and new transition variables in the trigger, a reference to a transition variable must be qualified with the associated correlation name.

The following example accesses a new transition table to capture the set of rows that are inserted into the INVOICE table:

```
CREATE TRIGGER LRG_ORDR
  AFTER INSERT ON INVOICE
  REFERENCING NEW TABLE AS N_TABLE
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    SELECT LARGE_ORDER_ALERT(CUST_NO,
    TOTAL_PRICE, DELIVERY_DATE)
    FROM N_TABLE WHERE TOTAL_PRICE > 10000;
  END
```

The SELECT statement in LRG_ORDER causes user-defined function LARGE_ORDER_ALERT to execute for each row in transition table N_TABLE that satisfies the WHERE clause (TOTAL_PRICE > 10000).

Triggered action:

When a trigger is activated, a triggered action occurs. Every trigger has one triggered action, which consists of an optional trigger condition and a trigger body.

Trigger condition:

If you want the triggered action to occur only when certain conditions are true, code a trigger condition. A trigger condition is similar to a predicate in a SELECT, except that the trigger condition begins with WHEN, rather than WHERE. If you do not include a trigger condition in your triggered action, the trigger body executes every time the trigger is activated.

For a row trigger, Db2 evaluates the trigger condition once for each modified row of the subject table. For a statement trigger, Db2 evaluates the trigger condition once for each execution of the triggering SQL statement.

If the trigger condition of a before trigger has a fullselect, the fullselect cannot reference the subject table.

The following example shows a trigger condition that causes the trigger body to execute only when the number of ordered items is greater than the number of available items:

```
CREATE TRIGGER CK_AVAIL
  NO CASCADE BEFORE INSERT ON ORDERS
  REFERENCING NEW AS NEW_ORDER
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_ORDER.QUANTITY >
    (SELECT ON_HAND FROM PARTS
     WHERE NEW_ORDER.PARTNO=PARTS.PARTNO))
     BEGIN ATOMIC
       VALUES(ORDER_ERROR(NEW_ORDER.PARTNO,
         NEW_ORDER.QUANTITY));
     END
```

Trigger body:

In the trigger body, you code the SQL statements that you want to execute whenever the trigger condition is true. If the trigger body consists of more than one statement, it must begin with BEGIN ATOMIC and end with END. You cannot include host variables or parameter markers in your trigger body.

The statements that you can use in a trigger body depend on the activation time of the trigger. For a list of valid SQL statements for triggers, see *Table 2. Allowable SQL statements* under *SQL-trigger-body* in the CREATE TRIGGER (Db2 SQL) topic.

Because you can include INSERT, DELETE (searched), UPDATE (searched), and MERGE statements in your trigger body, execution of the trigger body might cause activation of other triggers. See "Trigger cascading" on page 162 for more information.

Examples

Example 1

Define a trigger to increment the count of employees when a new employee is hired. The following example also explains how to determine why an SQL statement is allowed in the trigger.

```
    CREATE TRIGGER NEW_HIRE
       AFTER INSERT ON EMPLOYEE
```

```
    FOR EACH ROW MODE DB2SQL
    BEGIN ATOMIC
      1
      UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
    END
```

The UPDATE statement (1) is an SQL statement that is allowed because it is listed in *Table 2. Allowable SQL statements*.

The following list provides more detailed information about SQL statements that are valid in triggers:

- fullselect, CALL, and VALUES

Use a fullselect or the VALUES statement in a trigger body to conditionally or unconditionally invoke a user-defined function. Use the CALL statement to invoke a stored procedure. See "Invoking a stored procedure or user-defined function from a trigger" on page 158 for more information on invoking user-defined functions and stored procedures from triggers.

A fullselect in the trigger body of a before trigger cannot reference the subject table.

Example 2

Define a trigger to return an error condition and back out any changes that are made by the trigger, as well as actions that result from referential constraints on the subject table. Use the SIGNAL statement to indicate the error information to be returned. When Db2 executes the SIGNAL statement, it returns an SQLCA to the application with SQLCODE -438. The SQLCA also includes the following values, which you supply in the SIGNAL statement:

- A 5-character value that Db2 uses as the SQLSTATE
- An error message that Db2 places in the SQLERRMC field

In the following example, the SIGNAL statement causes Db2 to return an SQLCA with SQLSTATE 75001 and terminate the salary update operation if an employee's salary increase is over 20%:

```
CREATE TRIGGER SAL_ADJ
  BEFORE UPDATE OF SALARY ON EMP
  REFERENCING OLD AS OLD_EMP
  NEW AS NEW_EMP
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
  BEGIN ATOMIC
    SIGNAL SQLSTATE '75001'
      ('Invalid Salary Increase - Exceeds 20%');
  END
```

Example 3

Define a trigger to assign the current date to the HIRE_DATE column when a row is inserted into the EMP table. Because before triggers operate on rows of a table before those rows are modified, you cannot perform operations in the body of a before trigger that directly modify the subject table. You can, however, use the SET *transition-variable* statement to modify the values in a row before those values go into the table. For example, this trigger uses a new transition variable (NEW_VAR.HIRE_DATE) to assign today's date for the new employee's hire date:

```
CREATE TRIGGER HIREDATE
  NO CASCADE BEFORE INSERT ON EMP
  REFERENCING NEW AS NEW_VAR
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET NEW_VAR.HIRE_DATE = CURRENT_DATE;
  END
```

Example 4

In the following example, table CLASS_SCHED contains a row for the class schedule of each class at a school. When a class schedule row is added to the table, trigger VALIDATE_SCHED is activated. In the trigger, SQL control statements are used to check for and respond to the following errors in the class start and end times:

Related reference

CREATE TRIGGER (Db2 SQL)

LOAD (Db2 Utilities)

Invoking a stored procedure or user-defined function from a trigger

A trigger body can include only SQL statements. To perform actions or use logic that is not available in SQL statements, create user-defined functions or stored procedures. Then invoke them from within the trigger body.

About this task

Introductory concepts

Triggers (Introduction to Db2 for z/OS)

Restriction: You cannot include INSERT, UPDATE, DELETE, or MERGE statements in stored procedures or user-defined functions that are invoked by a BEFORE TRIGGER. These actions are not allowed, because BEFORE triggers must not modify any table.

Procedure

To invoke a stored procedure or user-defined function from a trigger:

1. Ensure that the stored procedure or user-defined function is defined before the trigger is defined.
   - Define procedures by using the CREATE PROCEDURE statement.
   - Define triggers by using the CREATE FUNCTION statement.
2. Invoke the user-defined function or stored procedure by performing one of the following actions:
   - To invoke a user-defined function, include the user-defined function in one of the following statements in the trigger:

     SELECT statement

     Use a SELECT statement to execute the function conditionally. The number of times that the user-defined function executes depends on the number of rows in the result table of the SELECT statement. For example, in the following trigger, the SELECT statement invokes user-defined function LARGE_ORDER_ALERT. This function executes once for each row in transition table N_TABLE with an order price of more than 10000:

     ```
     CREATE TRIGGER LRG_ORDR
       AFTER INSERT ON INVOICE
       REFERENCING NEW TABLE AS N_TABLE
       FOR EACH STATEMENT MODE DB2SQL
       BEGIN ATOMIC
         SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
           FROM N_TABLE WHERE TOTAL_PRICE > 10000;
       END
     ```

     VALUES statement

     Use the VALUES statement to execute a function unconditionally. The function executes once for each execution of a statement trigger or once for each row in a row trigger. In the following example, user-defined function PAYROLL_LOG executes every time the trigger PAYROLL1 is activated. This trigger is activated when an update operation occurs.

     ```
     CREATE TRIGGER PAYROLL1
       AFTER UPDATE ON PAYROLL
       FOR EACH STATEMENT MODE DB2SQL
       BEGIN ATOMIC
         VALUES(PAYROLL_LOG(USER, 'UPDATE',
           CURRENT TIME, CURRENT DATE));
       END
     ```

     ]
   - To invoke a stored procedure, include a CALL statement in the trigger. The parameters of this stored procedure call must be constants, transition variables, table locators, or expressions.

If the parameter is a transition variable or table locator, and the CALL statement is in a BEFORE or AFTER trigger, Db2 returns a warning.

3. To pass transition tables from the trigger to the user-defined function or stored procedure, use table locators.

   When you call a user-defined function or stored procedure from a trigger, you might want to give the function or procedure access to the entire set of modified rows. In this case, use table locators to pass a pointer to the old or new transition table.

   Most of the code for using a table locator is in the function or stored procedure that receives the locator.

   To pass the transition table from a trigger, specify the parameter TABLE *transition-table-name* when you invoke the function or stored procedure. This parameter causes Db2 to pass a table locator for the transition table to the user-defined function or stored procedure. For example, the following trigger passes a table locator for a transition table NEWEMPS to stored procedure CHECKEMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    CALL CHECKEMP(TABLE NEWEMPS);
  END
```

Related concepts

Steps to creating and using a user-defined function
A user-defined function is similar to a host language subprogram or function. However, a user-defined function is often the better choice for an SQL application because you can invoke it in an SQL statement.

Related tasks

Accessing transition tables in a user-defined function or stored procedure
If you want to refer to the entire set of rows that a triggering SQL statement modifies, rather than to individual rows, use a transition table. You can reference a transition table in user-defined functions and procedures that are invoked from a trigger.

Creating stored procedures
A *stored procedure* is executable code that can be called by other programs. The process for creating one depends on the type of procedure.

Creating a user-defined function
You can extend the SQL functionality of Db2 by adding your own or third party vendor function definitions.

Related reference

CALL (Db2 SQL)
CREATE FUNCTION (Db2 SQL)
CREATE PROCEDURE (Db2 SQL)
select-statement (Db2 SQL)
VALUES (Db2 SQL)

Inserting, updating, and deleting data in views by using INSTEAD OF triggers

INSTEAD OF triggers are triggers that execute instead of the INSERT, UPDATE, or DELETE statement that activates the trigger. You can define these triggers on views only. Use INSTEAD OF triggers to insert, update, and delete data in complex views.

About this task

Complex views are those views that are defined on expressions or multiple tables. In some cases, those views are read only. In these cases, INSTEAD OF triggers make the insert, update and delete operations possible. If the complex view is not read only, you can request an insert, update, or delete operation. However, Db2 automatically decides how to perform that operation on the base tables that are referenced in the view. With INSTEAD OF triggers, you can define exactly how Db2 is to execute an insert, update, or delete operation on the view. You no longer leave the decision to Db2.

Procedure

To insert, update, or delete data in a view by using INSTEAD OF triggers:

1. Define one or more INSTEAD OF triggers on the view by using a CREATE TRIGGER statement.

   You can create one trigger for each of the following operations: INSERT, UPDATE, and DELETE. These triggers define the action that Db2 is to take for each of these operations.
2. Submit a INSERT, UPDATE, or DELETE statement on the view.

   Db2 executes the appropriate INSTEAD OF trigger.

Example

Suppose that you create the following view on the sample tables DSN8B10.EMP and DSN8B10.DEPT:

```
CREATE VIEW EMPV (EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO, HIREDATE,DEPTNAME)
  AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO, HIREDATE, DEPTNAME
     FROM DSN8B10.EMP, DSN8B10.DEPT WHERE DSN8B10.EMP.WORKDEPT
          = DSN8B10.DEPT.DEPTNO
```

Suppose that you also define the following three INSTEAD OF triggers:

```
CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP
FOR EACH ROW MODE DB2SQL
  INSERT INTO DSN8B10.EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,
                           PHONENO, HIREDATE)
    VALUES(NEWEMP.EMPNO, NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
       COALESCE((SELECT D.DEPTNO FROM DSN8B10.DEPT AS D
                   WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                RAISE_ERROR('70001', 'Unknown department name')),
       NEWEMP.PHONENO, NEWEMP.HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  UPDATE DSN8B10.EMP AS E
   SET (E.FIRSTNME, E.MIDINIT, E.LASTNAME, E.WORKDEPT, E.PHONENO,
        E.HIREDATE)
       = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
       COALESCE((SELECT D.DEPTNO FROM DSN8B10.DEPT AS D
                  WHERE D.DEPTNAME = OLDEMP.DEPTNAME),
                RAISE_ERROR ('70001', 'Unknown department name'))
       NEWEMP.PHONENO, NEWEMP.HIREDATE)
   WHERE NEWEMP.EMPNO = E.EMPNO;
  UPDATE DSN8B10.DEPT D SET D.DEPTNAME=NEWEMP.DEPTNAME
   WHERE D.DEPTNAME=OLDEMP.DEPTNAME;
END

CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP
FOR EACH ROW MODE DB2SQL
  DELETE FROM DSN8B10.EMP AS E WHERE E.EMPNO = OLDEMP.EMPNO
```

Because the view is on a query with an inner join, the view is read only. However, the INSTEAD OF triggers makes insert, update, and delete operations possible.

The following table describes what happens for various insert, update, and delete operations on the EMPV view.

Table 42. Results of INSTEAD OF triggers

SQL statement	Result
`INSERT INTO EMPV VALUES (...)`	The EMPV_INSERT trigger is activated. This trigger inserts the row into the base table DSN8B10.EMP if the department name matches a value in the WORKDEPT column in the DSN8B10.DEPT table. Otherwise, an error is returned. If a query had been used instead of a VALUES clause on the INSERT statement, the trigger body would be processed for each row from the query.
`UPDATE EMPV SET DEPTNAME='PLANNING & STRATEGY' WHERE DEPTNAME='PLANNING'`	The EMPV_UPDATE trigger is activated. This trigger updates the DEPTNAME column in the DSN8B10.DEPT for the any qualifying rows.
`DELETE FROM EMPV WHERE HIREDATE<'1910-01-01'`	The EMPV_DELETE trigger is activated. This trigger deletes the qualifying rows from the DSN8B10.EMP table.

Related reference

CREATE TRIGGER (Db2 SQL)

Errors encountered in a trigger

An SQL statement in a trigger body may fail during trigger execution, causing an error to occur.

If any SQL statement in the trigger body fails during trigger execution, Db2 rolls back all changes that are made by the triggering SQL statement and the triggered SQL statements. However, if the trigger body executes actions that are outside of the control of Db2, or are not under the same commit coordination as the Db2 subsystem in which the trigger executes, Db2 cannot undo those actions. Examples of external actions that are not under the control of Db2 are:

- Performing updates that are not under RRS commit control
- Sending an electronic mail message

If the trigger executes external actions that are under the same commit coordination as the Db2 subsystem under which the trigger executes, and an error occurs during trigger execution, Db2 places the application process that issued the triggering statement in a must-rollback state. The application must then execute a rollback operation to roll back those external actions. Examples of external actions that are under the same commit coordination as the triggering SQL operation are:

- Executing a distributed update operation
- From a user-defined function or stored procedure, executing an external action that affects an external resource manager that is under RRS commit control.

Trigger packages

A *trigger package* is a special type of package that is created only when you execute a CREATE TRIGGER statement. A trigger package executes only when the associated trigger is activated.

As with any other package, Db2 marks a trigger package invalid when you drop a table, index, or view on which the trigger package depends. Db2 executes an automatic rebind the next time the trigger is activated. However, if the automatic rebind fails, Db2 does not mark the trigger package as inoperative.

Unlike other packages, a trigger package is freed if you drop the table on which the trigger is defined, so you can re-create the trigger package only by recreating the table and the trigger.

Use the subcommand REBIND TRIGGER PACKAGE to rebind the trigger package. You can also use REBIND TRIGGER PACKAGE to change the option values to change a limited subset of the default bind options that Db2 used when creating the package.

Related reference

REBIND TRIGGER PACKAGE (DSN) (Db2 Commands)

Trigger cascading

When a trigger performs an SQL operation, it might modify the subject table or other tables with triggers, therefore Db2 also activates those triggers. This situation is called trigger cascading.

A trigger that is activated as the result of another trigger can be activated at the same level as the original trigger or at a different level. Two triggers, A and B, are activated at different levels if trigger B is activated after trigger A is activated and completes before trigger A completes. If trigger B is activated after trigger A is activated and completes after trigger A completes, then the triggers are at the same level.

For example, in these cases, trigger A and trigger B are activated at the same level:

- Table X has two triggers that are defined on it, A and B. A is a before trigger and B is an after trigger. An update to table X causes both trigger A and trigger B to activate.
- Trigger A updates table X, which has a referential constraint with table Y, which has trigger B defined on it. The referential constraint causes table Y to be updated, which activates trigger B.

In these cases, trigger A and trigger B are activated at different levels:

- Trigger A is defined on table X, and trigger B is defined on table Y. Trigger B is an update trigger. An update to table X activates trigger A, which contains an UPDATE statement on table B in its trigger body. This UPDATE statement activates trigger B.
- Trigger A calls a stored procedure. The stored procedure contains an INSERT statement for table X, which has insert trigger B defined on it. When the INSERT statement on table X executes, trigger B is activated.

When triggers are activated at different levels, it is called *trigger cascading.* Trigger cascading can occur only for after triggers because Db2 does not support cascading of before triggers.

To prevent the possibility of endless trigger cascading, Db2 supports only 16 levels of cascading of triggers, stored procedures, and user-defined functions. If a trigger, user-defined function, or stored procedure at the 17th level is activated, Db2 returns SQLCODE -724 and backs out all SQL changes in the 16 levels of cascading. However, as with any other SQL error that occurs during trigger execution, if any action occurs that is outside the control of Db2, that action is not backed out.

You can write a monitor program that issues IFI READS requests to collect Db2 trace information about the levels of cascading of triggers, user-defined functions, and stored procedures in your programs.

Related tasks

Invoking IFI from a monitor program (Db2 Performance)

Activation order of multiple triggers

You can create multiple triggers for the same subject table, event, and activation time. The order in which those triggers are activated is the order in which the triggers were created.

Db2 records the timestamp when each CREATE TRIGGER statement executes. When an event occurs in a table that activates more than one trigger, Db2 uses the stored timestamps to determine which trigger to activate first.

Db2 always activates all before triggers that are defined on a table before the after triggers that are defined on that table, but within the set of before triggers, the activation order is by timestamp, and within the set of after triggers, the activation order is by timestamp.

In this example, triggers NEWHIRE1 and NEWHIRE2 have the same triggering event (INSERT), the same subject table (EMP), and the same activation time (AFTER). Suppose that the CREATE TRIGGER statement for NEWHIRE1 is run before the CREATE TRIGGER statement for NEWHIRE2:

```
CREATE TRIGGER NEWHIRE1
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
```

```
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEWHIRE2
  AFTER INSERT ON EMP
  REFERENCING NEW AS N_EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE DEPTS SET NBEMP = NBEMP + 1
      WHERE DEPT_ID = N_EMP.DEPT_ID;
  END
```

When an insert operation occurs on table EMP, Db2 activates NEWHIRE1 first because NEWHIRE1 was created first. Now suppose that someone drops and re-creates NEWHIRE1. NEWHIRE1 now has a later timestamp than NEWHIRE2, so the next time an insert operation occurs on EMP, NEWHIRE2 is activated before NEWHIRE1.

If two row triggers are defined for the same action, the trigger that was created earlier is activated first for all affected rows. Then the second trigger is activated for all affected rows. In the previous example, suppose that an INSERT statement with a fullselect inserts 10 rows into table EMP. NEWHIRE1 is activated for all 10 rows, then NEWHIRE2 is activated for all 10 rows.

Related reference

CREATE TRIGGER (Db2 SQL)

Interactions between triggers and referential constraints

When you create triggers, you need to understand the interactions among the triggers and constraints on your tables. You also need to understand the effect that the order of processing of those constraints and triggers can have on the results.

In general, the following steps occur when triggering SQL statement S1 performs an insert, update, or delete operation on table T1:

1. Db2 determines the rows of T1 to modify. Call that set of rows M1. The contents of M1 depend on the SQL operation:
   - For a delete operation, all rows that satisfy the search condition of the statement for a searched delete operation, or the current row for a positioned delete operation
   - For an insert operation, the row identified by the VALUES statement, or the rows identified by the result table of a SELECT clause within the INSERT statement
   - For an update operation, all rows that satisfy the search condition of the statement for a searched update operation, or the current row for a positioned update operation
2. Db2 processes all before triggers that are defined on T1, in order of creation.

   Each before trigger executes the triggered action once for each row in M1. If M1 is empty, the triggered action does not execute.

   If an error occurs when the triggered action executes, Db2 rolls back all changes that are made by S1.
3. Db2 makes the changes that are specified in statement S1 to table T1, unless an INSTEAD OF trigger is defined for that action. If an appropriate INSTEAD OF trigger is defined, Db2 executes the trigger instead of the statement and skips the remaining steps in this list.

   If an error occurs, Db2 rolls back all changes that are made by S1.
4. If M1 is not empty, Db2 applies all the following constraints and checks that are defined on table T1:
   - Referential constraints
   - Check constraints
   - Checks that are due to updates of the table through views defined WITH CHECK OPTION

   Application of referential constraints with rules of DELETE CASCADE or DELETE SET NULL are activated before delete triggers or before update triggers on the dependent tables.

If any constraint is violated, Db2 rolls back all changes that are made by constraint actions or by statement S1.

5. Db2 processes all after triggers that are defined on T1, and all after triggers on tables that are modified as the result of referential constraint actions, in order of creation.

   Each after row trigger executes the triggered action once for each row in M1. If M1 is empty, the triggered action does not execute.

   Each after statement trigger executes the triggered action once for each execution of S1, even if M1 is empty.

If any triggered actions contain SQL insert, update, or delete operations, Db2 repeats steps 1 through 5 for each operation.

If an error occurs when the triggered action executes, or if a triggered action is at the 17th level of trigger cascading, Db2 rolls back all changes that are made in step 5 and all previous steps.

For example, table DEPT is a parent table of EMP, with these conditions:

- The DEPTNO column of DEPT is the primary key.
- The WORKDEPT column of EMP is the foreign key.
- The constraint is ON DELETE SET NULL.

Suppose the following trigger is defined on EMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(CHECKEMP(TABLE NEWEMPS));
  END
```

Also suppose that an SQL statement deletes the row with department number E21 from DEPT. Because of the constraint, Db2 finds the rows in EMP with a WORKDEPT value of E21 and sets WORKDEPT in those rows to null. This is equivalent to an update operation on EMP, which has update trigger EMPRAISE. Therefore, because EMPRAISE is an after trigger, EMPRAISE is activated after the constraint action sets WORKDEPT values to null.

Interactions between triggers and tables that have multilevel security with row-level granularity

A BEFORE trigger affects the value of the transition variable that is associated with a security label column.

If a subject table has a security label column, the column in the transition table or transition variable that corresponds to the security label column in the subject table does not inherit the security label attribute. This means that the multilevel security check with row-level granularity is not enforced for the transition table or the transition variable. If you add a security label column to a subject table using the ALTER TABLE statement, the rules are the same as when you add any column to a subject table because the column in the transition table or the transition variable that corresponds to the security label column does not inherit the security label attribute.

If the ID you are using does not have write-down privilege and you execute an insert or update operation, the security label value of your ID is assigned to the security label column for the rows that you are inserting or updating.

When a BEFORE trigger is activated, the value of the transition variable that corresponds to the security label column is the security label of the ID if either of the following conditions is true:

- The user does not have write-down privilege
- The value for the security label column is not specified

If the user does not have write-down privilege, and the trigger changes the transition variable that corresponds to the security label column, the value of the security label column is changed back to the security label value of the user before the row is written to the page.

Related concepts

Multilevel security (Managing Security)

Triggers that return inconsistent results

When you create triggers and write SQL statements that activate those triggers, you need to ensure that executing those statements always produces the same results.

Two common reasons that you can get inconsistent results are:

- Positioned UPDATE or DELETE statements that use uncorrelated subqueries cause triggers to operate on a larger result table than you intended.
- Db2 does not always process rows in the same order, so triggers that propagate rows of a table can generate different result tables at different times.

The following examples demonstrate these situations.

Example: Effect of an uncorrelated subquery on a triggered action: Suppose that tables T1 and T2 look like this:

```
Table T1     Table T2
   A1           B1
   ==           ==
    1            1
    2            2
```

The following trigger is defined on T1:

```
CREATE TRIGGER TR1
  AFTER UPDATE OF T1
  FOR EACH ROW
  MODE DB2SQL
    BEGIN ATOMIC
      DELETE FROM T2 WHERE B1 = 2;
    END
```

Now suppose that an application executes the following statements to perform a positioned update operation:

```
EXEC SQL BEGIN DECLARE SECTION;
 long hv1;
EXEC SQL END DECLARE SECTION;
⋮
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT A1 FROM T1
  WHERE A1 IN (SELECT B1 FROM T2)
  FOR UPDATE OF A1;
⋮
EXEC SQL OPEN C1;
⋮
while(SQLCODE>=0 && SQLCODE!=100)
{
  EXEC SQL FETCH C1 INTO :hv1;
  UPDATE T1 SET A1=5 WHERE CURRENT OF C1;
}
```

When Db2 executes the FETCH statement that positions cursor C1 for the first time, Db2 evaluates the subselect, SELECT B1 FROM T2, to produce a result table that contains the two rows of column T2:

```
1
2
```

When Db2 executes the positioned UPDATE statement for the first time, trigger TR1 is activated. When the body of trigger TR1 executes, the row with value 2 is deleted from T2. However, because SELECT B1 FROM T2 is evaluated only once, when the FETCH statement is executed again, Db2 finds the second row of T1, even though the second row of T2 was deleted. The FETCH statement positions the cursor to the second row of T1, and the second row of T1 is updated. The update operation causes the trigger to be activated again, which causes Db2 to attempt to delete the second row of T2, even though that row was already deleted.

To avoid processing of the second row after it should have been deleted, use a correlated subquery in the cursor declaration:

```
DCL C1 CURSOR FOR
  SELECT A1 FROM T1 X
  WHERE EXISTS (SELECT B1 FROM T2 WHERE X.A1 = B1)
  FOR UPDATE OF A1;
```

In this case, the subquery, SELECT B1 FROM T2 WHERE X.A1 = B1, is evaluated for each FETCH statement. The first time that the FETCH statement executes, it positions the cursor to the first row of T1. The positioned UPDATE operation activates the trigger, which deletes the second row of T2. Therefore, when the FETCH statement executes again, no row is selected, so no update operation or triggered action occurs.

Example: Effect of row processing order on a triggered action: The following example shows how the order of processing rows can change the outcome of an after row trigger.

Suppose that tables T1, T2, and T3 look like this:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    1         (empty)      (empty)
    2
```

The following trigger is defined on T1:

```
CREATE TRIGGER TR1
  AFTER UPDATE ON T1
  REFERENCING NEW AS N
  FOR EACH ROW
  MODE DB2SQL
    BEGIN ATOMIC
      INSERT INTO T2 VALUES(N.C1);
      INSERT INTO T3 (SELECT B1 FROM T2);
    END
```

Now suppose that a program executes the following UPDATE statement:

```
UPDATE T1 SET A1 = A1 + 1;
```

The contents of tables T2 and T3 after the UPDATE statement executes depend on the order in which Db2 updates the rows of T1.

If Db2 updates the first row of T1 first, after the UPDATE statement and the trigger execute for the first time, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            2            2
    2
```

After the second row of T1 is updated, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            2            2
    3            3            2
                              3
```

However, if Db2 updates the second row of T1 first, after the UPDATE statement and the trigger execute for the first time, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    1            3            3
    3
```

After the first row of T1 is updated, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            3            3
    3            2            3
                              2
```

Sequence objects

A *sequence* is a user-defined object that generates a sequence of numeric values according to the specification with which the sequence was created. Sequences, unlike identity columns, are not associated with tables. Applications refer to a sequence object to get its current or next value.

The sequence of numeric values is generated in a monotonically ascending or descending order. The relationship between sequences and tables is controlled by the application, not by Db2.

Your application can reference a sequence object and coordinate the value as keys across multiple rows and tables. However, a table column that gets its values from a sequence object does not necessarily have unique values in that column. Even if the sequence object has been defined with the NO CYCLE clause, some other application might insert values into that table column other than values you obtain by referencing that sequence object.

Db2 always generates sequence numbers in order of request. However, in a data sharing group where the sequence values are cached by multiple Db2 members simultaneously, the sequence value assignments might not be in numeric order. Additionally, you might have gaps in sequence number values for the following reasons:

- If Db2 terminates abnormally before it assigns all the cached values
- If your application rolls back a transaction that increments the sequence
- If the statement containing NEXT VALUE fails after it increments the sequence

You create a sequence object with the CREATE SEQUENCE statement, alter it with the ALTER SEQUENCE statement, and drop it with the DROP SEQUENCE statement. You grant access to a sequence with the GRANT (privilege) ON SEQUENCE statement, and revoke access to the sequence with the REVOKE (privilege) ON SEQUENCE statement.

The values that Db2 generates for a sequence depend on how the sequence is created. The START WITH option determines the first value that Db2 generates. The values advance by the INCREMENT BY value in ascending or descending order.

The MINVALUE and MAXVALUE options determine the minimum and maximum values that Db2 generates. The CYCLE or NO CYCLE option determines whether Db2 wraps the generated values when it reaches the maximum value for an ascending sequence or the minimum value in a descending sequence.

Keys across multiple tables: You can use the same sequence number as a key value in two separate tables by first generating the sequence value with a NEXT VALUE expression to insert the first row in the first table. You can then reference this same sequence value with a PREVIOUS VALUE expression to insert the other rows in the second table.

Example: Suppose that an ORDERS table and an ORDER_ITEMS table are defined in the following way:

```
CREATE TABLE ORDERS
  (ORDERNO     INTEGER NOT NULL,
   ORDER_DATE  DATE DEFAULT,
   CUSTNO      SMALLINT
```

```
   PRIMARY KEY (ORDERNO));

CREATE TABLE ORDER_ITEMS
  (ORDERNO     INTEGER NOT NULL,
   PARTNO      INTEGER NOT NULL,
   QUANTITY    SMALLINT NOT NULL,
   PRIMARY KEY (ORDERNO,PARTNO),
   CONSTRAINT REF_ORDERNO FOREIGN KEY (ORDERNO)
     REFERENCES ORDERS (ORDERNO) ON DELETE CASCADE);
```

You create a sequence named ORDER_SEQ to use as key values for both the ORDERS and ORDER_ITEMS tables:

```
CREATE SEQUENCE ORDER_SEQ AS INTEGER
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 20;
```

You can then use the same sequence number as a primary key value for the ORDERS table and as part of the primary key value for the ORDER_ITEMS table:

```
INSERT INTO ORDERS (ORDERNO, CUSTNO)
  VALUES (NEXT VALUE FOR ORDER_SEQ, 12345);

INSERT INTO ORDER_ITEMS (ORDERNO, PARTNO, QUANTITY)
  VALUES (PREVIOUS VALUE FOR ORDER_SEQ, 987654, 2);
```

The NEXT VALUE expression in the first INSERT statement generates a sequence number value for the sequence object ORDER_SEQ. The PREVIOUS VALUE expression in the second INSERT statement retrieves that same value because it was the sequence number most recently generated for that sequence object within the current application process.

Db2 object relational extensions

With the object extensions of Db2, you can incorporate object-oriented concepts and methodologies into your relational database by extending Db2 with richer sets of data types and functions.

With those extensions, you can store instances of object-oriented data types in columns of tables and operate on them using functions in SQL statements. In addition, you can control the types of operations that users can perform on those data types.

The object extensions that Db2 provides are:

- Large objects (LOBs)

  The VARCHAR, VARGRAPHIC, and VARBINARY data types have a storage limit of 32 KB. Although this might be sufficient for small- to medium-size text data, applications often need to store large text documents. They might also need to store a wide variety of additional data types such as audio, video, drawings, mixed text and graphics, and images. Db2 provides three data types to store these data objects as strings of up to 2 GB - 1 in size. The three data types are binary large objects (BLOBs), character large objects (CLOBs), and double-byte character large objects (DBCLOBs).

  For a detailed discussion of LOBs, see “Storing LOB data in a table” on page 124 and Large objects (LOBs) (Db2 SQL).
- Distinct types

  A distinct type is a user-defined data type that shares its internal representation with a built-in data type but is considered to be a separate and incompatible type for semantic purposes. For example, you might want to define a picture type or an audio type, both of which have quite different semantics, but which use the built-in data type BLOB for their internal representation.

  For a detailed discussion of distinct types, see “Distinct types” on page 169.
- User-defined functions

The built-in functions that are supplied with Db2 are a useful set of functions, but they might not satisfy all of your requirements. For those cases, you can use user-defined functions. For example, a built-in function might perform a calculation you need, but the function does not accept the distinct types you want to pass to it. You can then define a function based on a built-in function, called a sourced user-defined function, that accepts your distinct types. You might need to perform another calculation in your SQL statements for which no built-in function exists. In that situation, you can define and write an SQL function or an external function.

For a detailed discussion of user-defined functions, see "Steps to creating and using a user-defined function" on page 183.

Creating a distinct type

Distinct types are useful when you want Db2 to handle certain data differently than other data of the same data type. For example, even though all currencies can be declared as type DECIMAL, you do not want euros to be compared to Japanese yen.

Procedure

Issue the CREATE DISTINCT TYPE statement.
For example, you can create distinct types for euros and yen by issuing the following SQL statements:

```
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2);
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2);
```

Related reference

CREATE TYPE (distinct) (Db2 SQL)

Distinct types

A *distinct type* is a user-defined data type that shares its internal representation with a built-in data type (its *source type*), but is considered to be a separate and incompatible data type for most operations.

Each distinct type has the same internal representation as a built-in data type.

Suppose you want to define some audio and video data in a Db2 table. You can define columns for both types of data as BLOB, but you might want to use a data type that more specifically describes the data. To do that, define distinct types. You can then use those types when you define columns in a table or manipulate the data in those columns. For example, you can define distinct types for the audio and video data like this:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M);
CREATE DISTINCT TYPE VIDEO AS BLOB (1M);
```

Then, your CREATE TABLE statement might look like this:

```
CREATE TABLE VIDEO_CATALOG;
  (VIDEO_NUMBER  CHAR(6)  NOT NULL,
   VIDEO_SOUND   AUDIO,
   VIDEO_PICS    VIDEO,
   ROW_ID        ROWID NOT NULL GENERATED ALWAYS);
```

For more information on LOB data, see "Storing LOB data in a table" on page 124 and Large objects (LOBs) (Db2 SQL).

After you define distinct types and columns of those types, you can use those data types in the same way you use built-in types. You can use the data types in assignments, comparisons, function invocations, and stored procedure calls. However, when you assign one column value to another or compare two column values, those values must be of the same distinct type. For example, you must assign a column value of type VIDEO to a column of type VIDEO, and you can compare a column value of type AUDIO only to a column of type AUDIO. When you assign a host variable value to a column with a distinct type, you can use any host data type that is compatible with the source data type of the distinct type. For example, to receive an AUDIO or VIDEO value, you can define a host variable like this:

```
SQL TYPE IS BLOB (1M) HVAV;
```

When you use a distinct type as an argument to a function, a version of that function that accepts that distinct type must exist. For example, if function SIZE takes a BLOB type as input, you cannot automatically use a value of type AUDIO as input. However, you can create a sourced user-defined function that takes the AUDIO type as input. For example:

```
CREATE FUNCTION SIZE(AUDIO)
  RETURNS INTEGER
  SOURCE SIZE(BLOB(1M));
```

Using distinct types in application programs: The main reason to use distinct types is because Db2 enforces *strong typing* for distinct types. Strong typing ensures that only functions, procedures, comparisons, and assignments that are defined for a data type can be used.

For example, if you have defined a user-defined function to convert U.S. dollars to euro currency, you do not want anyone to use this same user-defined function to convert Japanese yen to euros because the U.S. dollars to euros function returns the wrong amount. Suppose you define three distinct types:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL(9,2);
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2);
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2);
```

If a conversion function is defined that takes an input parameter of type US_DOLLAR as input, Db2 returns an error if you try to execute the function with an input parameter of type JAPANESE_YEN.

Example of distinct types, user-defined functions, and LOBs

You can create and use a distinct type based on a LOB data type.

The example in this topic demonstrates the following concepts:

- Creating a distinct type based on a LOB data type
- Defining a user-defined function with a distinct type as an argument
- Creating a table with a distinct type column that is based on a LOB type
- Defining a LOB table space, auxiliary table, and auxiliary index
- Inserting data from a host variable into a distinct type column based on a LOB column
- Executing a query that contains a user-defined function invocation
- Casting a LOB locator to the input data type of a user-defined function

Suppose that you keep electronic mail documents that are sent to your company in a Db2 table. The Db2 data type of an electronic mail document is a CLOB, but you define it as a distinct type so that you can control the types of operations that are performed on the electronic mail. The distinct type is defined like this:

```
CREATE DISTINCT TYPE E_MAIL AS CLOB(5M);
```

You have also defined and written user-defined functions to search for and return the following information about an electronic mail document:

- Subject
- Sender
- Date sent
- Message content
- Indicator of whether the document contains a user-specified string

The user-defined function definitions look like this:

```
CREATE FUNCTION SUBJECT(E_MAIL)
  RETURNS VARCHAR(200)
  EXTERNAL NAME 'SUBJECT'
```

```
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

```
CREATE FUNCTION SENDER(E_MAIL)
  RETURNS VARCHAR(200)
  EXTERNAL NAME 'SENDER'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

```
CREATE FUNCTION SENDING_DATE(E_MAIL)
  RETURNS DATE
  EXTERNAL NAME 'SENDDATE'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

```
CREATE FUNCTION CONTENTS(E_MAIL)
  RETURNS CLOB(1M)
  EXTERNAL NAME 'CONTENTS'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

```
CREATE FUNCTION CONTAINS(E_MAIL, VARCHAR (200))
  RETURNS INTEGER
  EXTERNAL NAME 'CONTAINS'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

The table that contains the electronic mail documents is defined like this:

```
CREATE TABLE DOCUMENTS
  (LAST_UPDATE_TIME  TIMESTAMP,
   DOC_ROWID      ROWID NOT NULL GENERATED ALWAYS,
   A_DOCUMENT        E_MAIL);
```

Because the table contains a column with a source data type of CLOB, the table requires an associated LOB table space, auxiliary table, and index on the auxiliary table. Use statements like this to define the LOB table space, the auxiliary table, and the index:

```
CREATE LOB TABLESPACE DOCTSLOB
  LOG YES
  GBPCACHE SYSTEM;

CREATE AUX TABLE DOCAUX_TABLE
  IN DOCTSLOB
  STORES DOCUMENTS COLUMN A_DOCUMENT;

CREATE INDEX A_IX_DOC ON DOCAUX_TABLE;
```

To populate the document table, you write code that executes an INSERT statement to put the first part of a document in the table, and then executes multiple UPDATE statements to concatenate the remaining parts of the document. For example:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hv_current_time[26];
  SQL TYPE IS CLOB (1M) hv_doc;
EXEC SQL END DECLARE SECTION;
/* Determine the current time and put this value */
/* into host variable hv_current_time.           */
```

```
/* Read up to 1 MB of document data from a file  */
/* into host variable hv_doc.                    */
⋮
/* Insert the time value and the first 1 MB of   */
/* document data into the table.                 */
EXEC SQL INSERT INTO DOCUMENTS
  VALUES(:hv_current_time, DEFAULT, E_MAIL(:hv_doc));

/* Although there is more document data in the   */
/* file, read up to 1 MB more of data, and then  */
/* use an UPDATE statement like this one to      */
/* concatenate the data in the host variable     */
/* to the existing data in the table.            */
EXEC SQL UPDATE DOCUMENTS
  SET A_DOCUMENT = A_DOCUMENT || E_MAIL(:hv_doc)
  WHERE LAST_UPDATE_TIME = :hv_current_time;
```

Now that the data is in the table, you can execute queries to learn more about the documents. For example, you can execute this query to determine which documents contain the word "performance":

```
SELECT SENDER(A_DOCUMENT), SENDING_DATE(A_DOCUMENT),
  SUBJECT(A_DOCUMENT)
  FROM DOCUMENTS
  WHERE CONTAINS(A_DOCUMENT,'performance') = 1;
```

Because the electronic mail documents can be very large, you might want to use LOB locators to manipulate the document data instead of fetching all of a document into a host variable. You can use a LOB locator on any distinct type that is defined on one of the LOB types. The following example shows how you can cast a LOB locator as a distinct type, and then use the result in a user-defined function that takes a distinct type as an argument:

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  SQL TYPE IS CLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION
⋮
/* Select a document into a CLOB locator.        */
EXEC SQL SELECT A_DOCUMENT, SUBJECT(A_DOCUMENT)
  INTO :hv_email_locator, :hv_subject
  FROM DOCUMENTS
  WHERE LAST_UPDATE_TIME = :hv_current_time;
⋮
/* Extract the subject from the document.  The   */
/* SUBJECT function takes an argument of type    */
/* E_MAIL, so cast the CLOB locator as E_MAIL.   */
EXEC SQL SET :hv_subject =
  SUBJECT(CAST(:hv_email_locator AS E_MAIL));
⋮
```

Arrays in SQL statements

An array is an ordered set of elements of a single built-in data type. An array can have an associated user-defined array type, or it can be the result of an SQL operation that returns an array value without an associated user-defined array type.

Arrays can be *ordinary arrays* and *associative arrays*.

Ordinary arrays have a user-defined upper bound. Elements in the array can be accessed and modified by their index value. Array elements are referenced in SQL statements by using one-based indexing; for example, MYARRAY[1], MYARRAY[2], and so on.

Associative arrays have no upper bound. Associative arrays contain an ordered set of zero or more elements, where each element in the array is ordered by and can be referenced by an associated index value. The data type of the index values can be an integer or a character string, but all index values for the array have the same data type.

Arrays can be used only in the following contexts:

- Parameters to SQL functions
- RETURN data types from SQL functions

- Parameters to SQL procedures
- SQL variables that are declared in SQL functions
- SQL variables that are declared in SQL procedures

You can create an array by creating an array type, and then defining an array variable of that type. For example:

```
-- CREATE ORDINARY ARRAY TYPE INTARRAY
CREATE TYPE INTARRAY AS INTEGER ARRAY[100];
-- IN AN SQL PROCEDURE, DEFINE ARRAY INTA OF THE INTARRAY TYPE
DECLARE INTA INTARRAY;
-- CREATE ASSOCIATIVE ARRAY TYPE CHARARRAY
CREATE TYPE CHARARRAY  AS CHAR(10) ARRAY[VARCHAR(10)];
-- IN AN SQL PROCEDURE, DEFINE ARRAY CHARA OF THE CHARARRAY TYPE
DECLARE CHARA CHARARRAY;
```

You cannot retrieve the contents of a column directly into an array. You need to use the ARRAY_AGG function to create an array that is the intermediate result of a SELECT statement, and then retrieve the contents of that array into an SQL array variable or parameter. For example:

```
-- INTB IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE INTARRAY.
-- COL2 IS AN INTEGER COLUMN.
-- ARRAY_AGG RETRIEVES THE VALUES FROM COL2, AND PUTS THEM INTO AN ARRAY.
SELECT ARRAY_AGG(COL2) INTO INTB FROM TABLE1;
```

You can retrieve data from an array by using the UNNEST specification to assign array elements to an intermediate result table. For example:

```
-- IDS AND NAMES ARE ARRAYS OF TYPE INTARRAY.
INSERT INTO PERSONS(ID, NAME)
  (SELECT T.I, T.N FROM UNNEST(IDS, NAMES) AS T(I, N));
```

To populate arrays, you use *array constructors*.

For example, this statement populates an ordinary array:

```
SET CHARA = ARRAY['1','2','3','4','5','6'];
```

For example, these statements populate an associative array, which must be populated one element at a time:

```
SET CANADACAPITALS['Alberta'] = 'Edmonton';
SET CANADACAPITALS['Manitoba'] = 'Winnipeg';
SET CANADACAPITALS['Ontario'] = 'Toronto';
SET CANADACAPITALS['Nova Scotia'] = 'Halifax';
```

A number of built-in functions are available for manipulating arrays. They are:

ARRAY_DELETE
Deletes elements from an array.

ARRAY_FIRST
Returns the minimum array index value of an array.

ARRAY_LAST
Returns the maximum array index value of an array.

ARRAY_NEXT
Returns the next larger array index value, relative to a specified array index value.

ARRAY_PRIOR
Returns the next smaller array index value, relative to a specified array index value.

CARDINALITY
Returns the number of elements in an array.

MAX_CARDINALITY
Returns the maximum number of elements that an array can contain.

TRIM_ARRAY
Deletes elements from the end of an ordinary array.

Related concepts

Array type comparisons (Db2 SQL)

Array type assignments (Db2 SQL)

Array types (Db2 SQL)

Related reference

Array constructor (Db2 SQL)

ARRAY_AGG (Db2 SQL)

ARRAY_DELETE (Db2 SQL)

ARRAY_FIRST (Db2 SQL)

ARRAY_NEXT (Db2 SQL)

ARRAY_PRIOR (Db2 SQL)

CARDINALITY (Db2 SQL)

MAX_CARDINALITY (Db2 SQL)

TRIM_ARRAY (Db2 SQL)

Example of using arrays in an SQL procedure

An example demonstrates many of the ways that you can use arrays in a native SQL procedure.

The example demonstrates how to:

- Create an associative array type.
- Create an ordinary array type.
- Create a stored procedure with arrays as parameters.
- Define arrays as SQL variables.
- Use the ARRAY_AGG built-in function in a cursor declaration, to assign the rows of a single-column result table to elements of an array. Use the cursor to retrieve the array into an SQL out parameter.
- Use an array constructor to initialize an array.
- Assign a constant or an expression to an array element.
- Use the UNNEST specification to generate the intermediate result table from an array for a subselect within an INSERT statement.
- Use the ARRAY_AGG built-in function to assign the rows of a single column result table to elements of an array, and then assign that array to an array SQL OUT parameter.
- Use the CARDINALITY built-in function to determine how many times to execute a WHILE loop.
- Use a parameter marker for an array variable and an array index in the WHERE clause of a SELECT statement.
- Use the ARRAY_AGG built-in function in the SELECT list of a SELECT INTO statement, and assign the resulting array to an array SQL OUT parameter.
- Update column values with array elements.

In this example, the pound sign (#) is used as the SQL terminator character.

```
--
-- CREATE ASSOCIATIVE ARRAY TYPES
--
CREATE TYPE CHARARRAY  AS CHAR(10) ARRAY[VARCHAR(3)]#
CREATE TYPE BIGINTARRAY AS BIGINT ARRAY[INTEGER]#
--
-- CREATE ORDINARY ARRAY TYPES
--
CREATE TYPE INTARRAY AS INTEGER ARRAY[100]#
CREATE TYPE STRINGARRAY AS VARCHAR(10) ARRAY[100]#
--
-- CREATE TABLES THAT ARE USED IN SQL PROCEDURE PROCESSPERSONS
--
```

```
CREATE TABLE PERSONS (ID INTEGER, NAME VARCHAR(10))#
CREATE TABLE ARRAYTEST (CHARCOL CHAR(10), INTCOL INT)#
-- SQL PROCEDURE PROCESSPERSONS HAS THREE ARRAY PARAMETERS:
-- OUTSETARRAY IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE STRINGARRAY.
-- OUTSELECTWITHCURSOR IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE STRINGARRAY.
-- OUTSELECTWITHARRAYAGG IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE INTARRAY.
--
CREATE PROCEDURE PROCESSPERSONS(OUT OUTSETARRAY STRINGARRAY,
                                INOUT INT0 INT,
                                OUT OUTSELECTWITHCURSOR STRINGARRAY,
                    OUT OUTMAXCARDINALITY BIGINT,
                    OUT OUTSELECTWITHARRAYAGG INTARRAY)
ARRAYDEMO: BEGIN
-- DECLARE SQL VARIABLES OF ORDINARY ARRAY TYPES
 DECLARE IDS_ORDARRAYVAR INTARRAY;
 DECLARE INT_ORDARRAYVAR INTARRAY;
 DECLARE NAMES_ORDARRAYVAR STRINGARRAY;
-- DECLARE SQL VARIABLES OF ASSOCIATIVE ARRAY TYPES
 DECLARE CHAR_ASSOCARRAYVAR CHARARRAY;
 DECLARE BIGINT_ASSOCARRAYVAR BIGINTARRAY;
-- DECLARE SCALAR SQL VARIABLES
 DECLARE DECFLOAT_VAR DECFLOAT;
 DECLARE BIGINT_VAR BIGINT;
 DECLARE SMALLINT_VAR SMALLINT;
 DECLARE INT_VAR INT DEFAULT 1;
 DECLARE STMT_VAR CHAR(100);
-- DECLARE A CURSOR
 DECLARE C2 CURSOR FOR S1;
--
-- THE RESULT TABLE OF CURSOR C1 IS AN ARRAY THAT IS POPULATED BY
-- RETRIEVING THE VALUES OF THE NAME COLUMN FROM TABLE PERSONS,
-- ORDERING THE VALUES BY ID, AND USING THE ARRAY_AGG FUNCTION
-- TO ASSIGN THE VALUES TO AN ARRAY.
--
 DECLARE C1 CURSOR FOR SELECT ARRAY_AGG(NAME ORDER BY ID) FROM PERSONS
   WHERE  NAME LIKE 'J%';
--
-- USE ARRAY CONSTRUCTORS TO INITIALIZE ARRAYS
--
 SET IDS_ORDARRAYVAR = ARRAY[5,6,7];
 SET NAMES_ORDARRAYVAR = ARRAY['BOB', 'ANN', 'SUE'];
 SET CHAR_ASSOCARRAYVAR['001']='1';
 SET CHAR_ASSOCARRAYVAR['002']='2';
 SET CHAR_ASSOCARRAYVAR['003']='3';
 SET CHAR_ASSOCARRAYVAR['004']='4';
 SET CHAR_ASSOCARRAYVAR['005']='5';
 SET CHAR_ASSOCARRAYVAR['006']='6';
 SET INT_ORDARRAYVAR = ARRAY[1,INTEGER(2),3+0,4,5,6] ;
 SET BIGINT_ASSOCARRAYVAR[1] = 9;
 SET BIGINT_ASSOCARRAYVAR[3] = 10;
 SET BIGINT_ASSOCARRAYVAR[5] = 11;
 SET BIGINT_ASSOCARRAYVAR[7] = 12;
 SET BIGINT_ASSOCARRAYVAR[9] = 13;
--
-- ASSIGN A CONSTANT TO AN ARRAY ELEMENT.
--
 SET IDS_ORDARRAYVAR[4] = 8;
--
-- ASSIGN AN EXPRESSION TO AN ARRAY ELEMENT.
--
 SET IDS_ORDARRAYVAR[5] = 8 * 4 ;
--
-- ASSIGN AN ARRAY ELEMENT TO ANOTHER ARRAY ELEMENT. USE AN EXPRESSION
-- TO IDENTIFY THE TARGET ARRAY ELEMENT.
--
 SET NAMES_ORDARRAYVAR[1+INT_VAR] = NAMES_ORDARRAYVAR[5] ;
--
-- POPULATE THE PERSONS TABLE WITH AN INSERT STATEMENT WITH A SUBSELECT:
-- - USE UNNEST TO RETRIEVE VALUES FROM AN ARRAY INTO AN INTERMEDIATE RESULT
--   TABLE.
-- - INSERT THE VALUES FROM THE INTERMEDIATE RESULT TABLE INTO
--   THE PERSONS TABLE.
--
 INSERT INTO PERSONS(ID, NAME)
  (SELECT T.I, T.N FROM UNNEST(IDS_ORDARRAYVAR, NAMES_ORDARRAYVAR) AS T(I, N));
--
-- USE THE ARRAY_AGG FUNCTION TO CREATE AN ARRAY FROM THE RESULT
-- TABLE OF A SELECT. THEN ASSIGN THAT ARRAY TO AN SQL OUT PARAMETER.
--
 SET OUTSETARRAY = (SELECT ARRAY_AGG(NAME ORDER BY ID)
  FROM PERSONS
  WHERE NAME LIKE '%O%');
```

```
--
-- USE THE CARDINALITY FUNCTION TO CONTROL THE NUMBER OF TIMES THAT
-- AN INSERT STATEMENT IS EXECUTED TO POPULATE TABLE ARRAYTEST
-- WITH ARRAY ELEMENTS.
--
 SET SMALLINT_VAR = 1;
 WHILE SMALLINT_VAR <= CARDINALITY(INT_ORDARRAYVAR) DO
  INSERT INTO ARRAYTEST VALUES
   (CHAR_ASSOCARRAYVAR[SMALLINT_VAR],
   INT_ORDARRAYVAR[SMALLINT_VAR]);
  SET SMALLINT_VAR = SMALLINT_VAR+1;
 END WHILE;
--
-- DYNAMICALLY EXECUTE AN SQL SELECT STATEMENT WITH A PARAMETER MARKER
-- FOR AN ARRAY, AND A PARAMETER MARKER FOR THE ARRAY INDEX.
--
 SET INT_VAR = 3;
 SET STMT_VAR =
  'SELECT INTCOL FROM ARRAYTEST WHERE INTCOL = ' ||
  'CAST(? AS INTARRAY)[?]';
 PREPARE S1 FROM STMT_VAR;
 OPEN C2 USING INT_ORDARRAYVAR, INT_VAR;
 FETCH C2 INTO INT0;
 CLOSE C2;
--
-- USE A CURSOR TO FETCH AN ARRAY THAT IS CREATED WITH THE ARRAY_AGG FUNCTION
-- INTO AN ARRAY SQL OUT PARAMETER.
--
 OPEN C1;
 FETCH C1 INTO OUTSELECTWITHCURSOR;
 CLOSE C1;
--
-- RETURN THE MAXIMUM CARDINALITY OF AN ARRAY USING THE MAX_CARDINALITY
-- FUNCTION, AND STORE THE VALUE IN AN SQL VARIABLE.
--
 SET OUTMAXCARDINALITY  = MAX_CARDINALITY(INT_ORDARRAYVAR);
--
-- IN A SELECT INTO STATEMENT, USE THE ARRAY_AGG FUNCTION TO
-- ASSIGN THE VALUES OF COLUMN INTCOL TO ARRAY ELEMENTS, AND ASSIGN
-- THOSE ELEMENTS TO ARRAY OUT PARAMETER OUTSELECTWITHARRAYAGG.
--
 SELECT ARRAY_AGG(INTCOL) INTO OUTSELECTWITHARRAYAGG FROM ARRAYTEST;
--
-- IN AN UPDATE STATEMENT, ASSIGN ARRAY ELEMENTS TO COLUMNS.
--
 SET SMALLINT_VAR = 1;
 WHILE SMALLINT_VAR <= CARDINALITY(INT_ORDARRAYVAR) DO
  UPDATE ARRAYTEST
   SET CHARCOL =
    CHAR_ASSOCARRAYVAR[SMALLINT_VAR], INTCOL = INT_ORDARRAYVAR[SMALLINT_VAR];
  SET SMALLINT_VAR = SMALLINT_VAR +1;
 END WHILE;
END#
```

Related concepts

Array type comparisons (Db2 SQL)

Array type assignments (Db2 SQL)

Related reference

Array constructor (Db2 SQL)

ARRAY_AGG (Db2 SQL)

CARDINALITY (Db2 SQL)

MAX_CARDINALITY (Db2 SQL)

Creating a user-defined function

You can extend the SQL functionality of Db2 by adding your own or third party vendor function definitions.

Before you begin

Set up the environment for user-defined functions, as described in Installation step 19: Configure Db2 for running stored procedures and user-defined functions (Db2 Installation and Migration).

About this task

A *user-defined function* is a small program that you can write to perform an operation, similar to a host language subprogram or function. However, a user-defined function is often the better choice for an SQL application because you can invoke it in an SQL statement. User-defined functions are created using the CREATE FUNCTION statement and registered to Db2 in the catalog.

A user-defined function is denoted by a function name followed by zero or more operands that are enclosed in parentheses. Like a built-in function, a user-defined function represents a relationship between a set of input values and a set of result values. The input values to a function are called *parameters* in the function definition. The input values to a function are called *arguments* when the function is invoked. For example, a function can be passed with two input arguments that have date and time data types and return a value with a timestamp data type as the result.

You can create several different types of user-defined functions, including external, SQL, and sourced user-defined functions. User-defined functions can also be categorized as scalar functions, which return a single value, or table functions, which return a table. Specifically, you can create the following types of user-defined functions:

External scalar
: The function is written in a programming language and returns a scalar value. The external executable routine (package) is registered with a database server along with various attributes of the function. Each time that the function is invoked, the package executes one or more times. See CREATE FUNCTION (external scalar) (Db2 SQL).

External table
: The function is written in a programming language. It returns a table to the subselect from which it was started by returning one row at a time, each time that the function is started. The external executable routine (package) is registered with a database server along with various attributes of the function. Each time that the function is invoked, the package executes one or more times. See CREATE FUNCTION (external table) (Db2 SQL).

Sourced
: The function is implemented by invoking another function (either built-in, external, SQL, or sourced) that exists at the server. The function inherits the attributes of the underlying source function. A sourced function does not have an associated package. See CREATE FUNCTION (sourced) (Db2 SQL).

SQL scalar
: The function is written exclusively in SQL statements and returns a scalar value. The body of an SQL scalar function is written in the SQL procedural language (SQL PL). The function is defined at the current server along with various attributes of the function.

Db2 supports two types of SQL scalar functions, inlined and compiled:

- *Inlined SQL scalar functions* contain a single RETURN statement, which returns the value of a simple expression. The function is not invoked as part of a query; instead, the *expression* in the RETURN statement of the function is copied (inlined) into the query itself. Therefore, a package is not generated for an inlined SQL scalar function.
- *Compiled SQL scalar functions* support a larger set of functionality, including all of the SQL PL statements. A package is generated for a compiled SQL scalar function. It contains the body of the function, including control statements. It might also contain statements generated by Db2. Each time that the function is invoked, the package executes one or more times.

When a CREATE FUNCTION statement for an SQL scalar function is processed, Db2 attempts to create an inlined SQL scalar function. If the function cannot be created as an inlined function, Db2 attempts to create a compiled SQL scalar function. For more information on the syntax and rules for these types of functions, see CREATE FUNCTION (inlined SQL scalar) (Db2 SQL) and CREATE FUNCTION (compiled SQL scalar) (Db2 SQL).

To determine what type of SQL scalar function is created, refer to the INLINE column of the SYSIBM.SYSROUTINES catalog table.

SQL table

The function is written exclusively as an SQL RETURN statement and returns a set of rows. The body of an SQL table function is written in the SQL procedural language. The function is defined at the current server along with various attributes. The function is not invoked as part of a query. Instead, the *expression* in the RETURN statement of the function is copied (inlined) into the query itself. Therefore, a package is not generated for an SQL table function. See CREATE FUNCTION (SQL table) (Db2 SQL).

The environment for user-defined functions includes application address space, from which a program invokes a user-defined function; a Db2 system, where the packages from the user-defined function are run; and a WLM-established address space, where the user-defined function may be executed; as shown in the following figure.

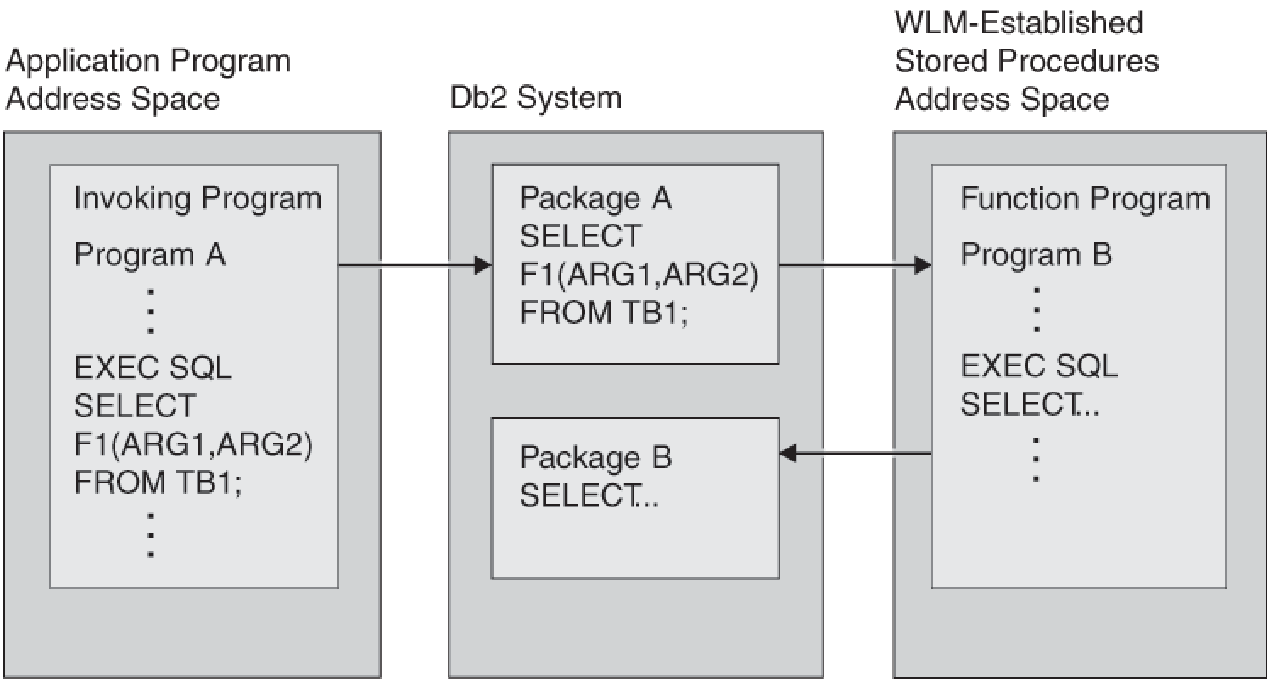


Figure 6. The user-defined function environment

For information on Java user-defined functions, see Java stored procedures and user-defined functions (Db2 Application Programming for Java). For user-defined functions in other languages, see the following instructions.

Procedure

To create a user-defined function:

1. Write and prepare the user-defined function, as described in "Writing an external user-defined function" on page 184.

   This step is necessary only for an external user-defined function.
2. Define the user-defined function to Db2 by issuing a CREATE FUNCTION statement that specifies the type of function that you want to create.

   For more information, see CREATE FUNCTION (Db2 SQL).
3. Invoke the user-defined function from an SQL application, as described in "Invoking a user-defined function" on page 451.

Definition for an SQL user-defined scalar function

You can define an SQL user-defined function to calculate the tangent of a value by using the existing built-in SIN and COS functions:

```
CREATE FUNCTION TAN (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  DETERMINISTIC
  RETURN SIN(X)/COS(X);
```

The logic of the function is contained in the function definition as the following statement:

```
RETURN SIN(X)/COS(X)
```

What to do next

If you discover after you define the function that you need to change a part of the definition, you can use an ALTER FUNCTION statement to change the definition. You cannot use ALTER FUNCTION to change some of the characteristics of a user-defined function definition.

Related concepts

Sample user-defined functions (Db2 SQL)

Related tasks

Controlling user-defined functions (Db2 Administration Guide)

Related reference

CREATE FUNCTION (Db2 SQL)

External functions

An *external user-defined function* is a function that is written in a programming language. An external function is defined to the database with a reference to an external program that contains the logic that is executed when the function is invoked.

An external user-defined function that returns a single value is a scalar function. An external user-defined function that returns a table is a table function.

You can write an external user-defined function in assembler, C, C++, COBOL, PL/I, or Java. User-defined functions that are written in COBOL can include object-oriented extensions, just as other Db2 COBOL programs can. User-defined functions that are written in Java follow coding guidelines and restrictions specific to Java. For information about writing Java user-defined functions, see Java stored procedures and user-defined functions (Db2 Application Programming for Java).

Examples

Example 1: Definition for an external user-defined scalar function

A programmer develops a user-defined function that searches for a string of maximum length 200 in a CLOB value whose maximum length is 500 KB. This CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRINGCLOB
  EXTERNAL NAME 'FINDSTR'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

The function returns a status code as an integer. The CAST FROM clause is specified because the function operation results in a floating point value, and users are expecting an integer result for their SQL statements. The user-defined function is written in C and contains no SQL statements.

Suppose that you want a FINDSTRING user-defined function to work on BLOB data types, as well as CLOB types. You can define another instance of a FINDSTRING user-defined function that specifies a BLOB type as input:

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRINGBLOB
  EXTERNAL NAME 'FNDBLOB'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC;
```

Each instance of FINDSTRING uses a different application program to implement the logic for the user-defined function.

Example 2: Definition for an external user-defined scalar function

A programmer has written a user-defined function for division. That is, this user-defined function is invoked when an application program executes a statement using the division operator (/), such as the following statement:

```
UPDATE TABLE1 SET INTCOL1="/"(INTCOL2,INTCOL3);
```

The user-defined function takes two integer values as input. The output from the user-defined function is of type integer. The user-defined function is in the MATH schema, is written in assembler, and contains no SQL statements. This CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION MATH."/" (INT, INT)
  RETURNS INTEGER
  SPECIFIC DIVIDE
  EXTERNAL NAME 'DIVIDE'
  LANGUAGE ASSEMBLE
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC;
```

Example 3: Definition for an external user-defined table function

An application programmer develops a user-defined function that receives two input values and returns a table. The two input values are:

- A character string of maximum length 30 that describes a subject
- A character string of maximum length 255 that contains text to search for

The user-defined function scans documents on the subject for the search string and returns a list of documents that match the search criteria, with an abstract for each document. The list is in the form of a two-column table. The first column is a character column of length 16 that contains document IDs. The second column is a varying-character column of maximum length 5000 that contains document abstracts.

The user-defined function is written in COBOL, uses SQL only to perform queries, and always produces the same output for given input. The CARDINALITY option specifies that you should expect an invocation of the user-defined function to return about 20 rows.

The following CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16), DOC_ABSTRACT VARCHAR(5000))
  EXTERNAL NAME 'DOCMTCH'
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  READS SQL DATA
```

```
    DETERMINISTIC
    CARDINALITY 20;
```

SQL scalar functions

An *SQL scalar function* is a user-defined function written in SQL and it returns a single value each time it is invoked. SQL scalar functions contain the source code for the user-defined function in the user-defined function definition. There are two kinds of SQL scalar functions, inlined and compiled.

All SQL scalar functions that were created prior to DB2 10 are inlined SQL scalar functions. Beginning with DB2 10, SQL scalar functions may be created as either inlined or compiled.

Db2 determines whether an SQL scalar function is inlined or compiled according to whether or not the CREATE FUNCTION statement that defines the function makes use of enhanced features. See CREATE FUNCTION (inlined SQL scalar) (Db2 SQL) and CREATE FUNCTION (compiled SQL scalar) (Db2 SQL) for more information.

An inlined SQL scalar function has a body with a single RETURN statement. The RETURN statement can return either a NULL value or a simple expression that does not reference a scalar fullselect. No package will be generated for an inlined SQL scalar function. During the preparation of an SQL statement that references the function (when the function is invoked), the expression specified in the RETURN statement of the function is simply inlined into that SQL statement.

A compiled SQL scalar function can have a body with logic written in SQL PL language. It can make use of any of the enhanced features for the CREATE FUNCTION statement including the support for TABLE LOCATOR data type for parameters, various options, and an enhanced RETURN statement that allows reference to a scalar fullselect. A package is created for a compiled SQL scalar function.

Compiled SQL scalar functions include support for versions and source code management. You can use compiled SQL scalar functions for the following tasks:

- Define multiple versions of an SQL scalar function, where one version is considered the "active" version.
- Activate a particular version of an SQL scalar function.
- Alter the routine options that are associated with a version of an SQL scalar function.
- Define a new version of an SQL scalar function by specifying the same function signature as the current version, and different routine options and function body.
- Replace the definition of an existing version by specifying the same function signature as the current version, and different routine options and function body.
- Drop a version of an SQL scalar function.
- Fall back to a previous version without requiring an explicit rebind or recompile, by activating the previous version.

You can deploy compiled SQL scalar functions to multiple servers to allow a wider community to use functions that have been thoroughly tested, without the risk of changing the logic in the routine body. Use the Unified Debugger to remotely debug compiled SQL scalar functions that execute on Db2 for z/OS servers.

To prepare an SQL scalar function for execution, you execute the CREATE FUNCTION statement, either statically or dynamically.

Example: Definition for a compiled SQL scalar user-defined function

The following example defines a scalar function that returns the text of an input string, in reverse order. The example also explains how to determine why various SQL statements are allowed in a compiled SQL scalar function.

A compiled SQL scalar CREATE FUNCTION statement contains an *SQL-routine-body*, as defined in CREATE FUNCTION (compiled SQL scalar) (Db2 SQL). The syntax diagram for *SQL-routine-body* defines the function body as a single SQL control statement. The syntax diagram for *SQL-control-statement* in SQL procedural language (SQL PL) (Db2 SQL) identifies the control statements that can be specified, including a RETURN statement.

An SQL function can contain multiple SQL statements if the outermost SQL statement is an *SQL-control-statement* that includes other SQL statements. These statements are defined as SQL procedure statements. The syntax diagram in SQL-procedure-statement (Db2 SQL) identifies the SQL statements that can be specified within a control statement. The syntax notes for *SQL-procedure-statement* clarify the SQL statements that are allowed in an SQL function.

```
     CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
       RETURNS VARCHAR(4000)
       DETERMINISTIC NO EXTERNAL ACTION
       CONTAINS SQL
       BEGIN  A
       DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT ''; B
       DECLARE LEN INT;  B
       IF INSTR IS NULL THEN  C
       RETURN NULL;  D
       END IF;
       SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));  E
       WHILE LEN > 0 DO  F
       SET (REVSTR, RESTSTR, LEN)  E
         = (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR,
         SUBSTR(RESTSTR, 2, LEN - 1),
         LEN - 1);
      END WHILE;
      RETURN REVSTR;  D
    END#  A
```

The SQL function has the following keywords and statements:

- The BEGIN and END keywords (A) indicate the beginning and the end of a compound statement.
- The DECLARE statements (B) are components of a compound statement, and define SQL variables within the compound statement. For more information on compound statements, see compound-statement (Db2 SQL).
- The IF statement (C), the RETURN statements (D), and the WHILE statement (F) are SQL control statements.
- The SET assignment statements (E) are SQL control statements that assign values to SQL variables.

SQL variables can be referenced anywhere in the compound statement in which they are declared, including any SQL statement that is directly or indirectly nested within that compound statement. See References to SQL parameters and variables (Db2 SQL) for more information.

Related tasks

Creating a user-defined function
You can extend the SQL functionality of Db2 by adding your own or third party vendor function definitions.

Related reference

CREATE FUNCTION (Db2 SQL)

SQL table functions

An *SQL table function* is a function that is written exclusively in SQL statements and returns a single result table.

An SQL table function can define a parameter as a distinct type, define a parameter for a transition table (for example, the TABLE LIKE ... AS LOCATOR syntax), and include a single SQL PL RETURN statement that returns a result table

The CREATE statement for an SQL table function is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

The ALTER statement for an SQL table function can be embedded in an application program or issued interactively. The ALTER statement is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Sourced functions

A *sourced function* is a function that invokes another function that already exists at the server. The function inherits the attributes of the underlying source function. The source function can be built-in, external, SQL, or sourced. Sourced functions can be used to extend built-in aggregate and scalar functions for use on distinct types.

You can use sourced functions to build upon existing built-in functions or other user-defined functions. Sourced functions are useful to extend built-in aggregate and scalar functions for use on distinct types.

To implement a sourced function, issue a CREATE FUNCTION statement and identify the function upon which you want to base the sourced function in the SOURCE clause.

Example: Definition of a sourced user-defined function

Suppose you need a user-defined function that finds a string in a value with a distinct type of BOAT. BOAT is a distinct type based on a BLOB data type. User-defined function FINDSTRINGBLOB has already been defined to take a BLOB data type as input and perform the required function, but it cannot be invoked with a value of the BOAT data type. The specific name for FINDSTRING is FINDSTRINGBLOB.

You can define a sourced user-defined function based on FINDSTRING to do the string search on values of type BOAT. Db2 implicitly casts the BOAT argument to a BLOB when the source function, FINDSTRING that accepts a BLOB value, is invoked. This CREATE FUNCTION statement defines the sourced user-defined function:

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
  RETURNS INTEGER
  SPECIFIC FINDSTRINGBOAT
  SOURCE SPECIFIC FINDSTRINGBLOB;
```

Related reference

CREATE FUNCTION (sourced) (Db2 SQL)

Steps to creating and using a user-defined function

A user-defined function is similar to a host language subprogram or function. However, a user-defined function is often the better choice for an SQL application because you can invoke it in an SQL statement.

This section contains information that applies to all user-defined functions and specific information about user-defined functions in languages other than Java.

Creating and using a user-defined function involves these steps:

- Setting up the environment for user-defined functions

  A systems administrator probably performs this step. The user-defined function environment is shown in the following figure.

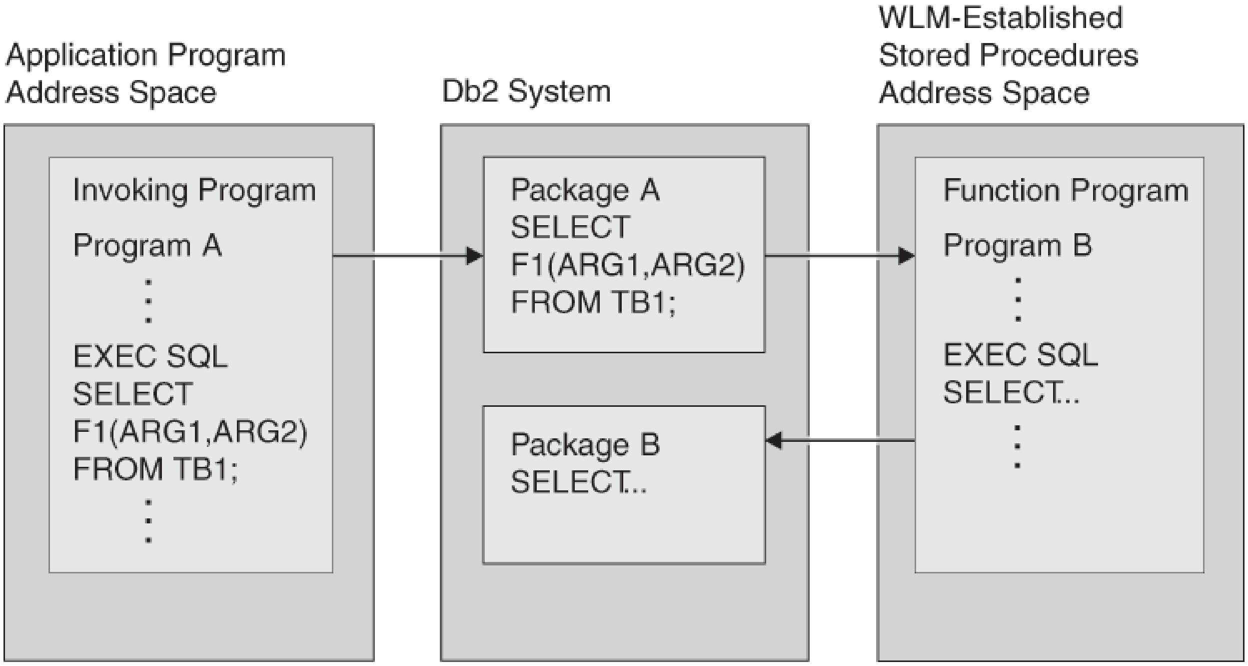


Figure 7. The user-defined function environment

It contains an application address space, from which a program invokes a user-defined function; a Db2 system, where the packages from the user-defined function are run; and a WLM-established address space, where the user-defined function is executed. The steps for setting up and maintaining the user-defined function environment are the same as for setting up and maintaining the environment for stored procedures in WLM-established address spaces.

- Writing and preparing the user-defined function

  This step is necessary only for an external user-defined function.

  The person who performs this step is called the user-defined function *implementer*.
- Defining the user-defined function to Db2

  The person who performs this step is called the user-defined function *definer*.
- Invoking the user-defined function from an SQL application

  The person who performs this step is called the user-defined function *invoker*.

Related concepts

Java stored procedures and user-defined functions (Db2 Application Programming for Java)

Writing an external user-defined function

An *external user-defined function* is written in a programming language and is similar to other SQL programs. You can include static or dynamic SQL statements, IFI calls, and Db2 commands that are issued through IFI calls.

Procedure

You can write an external user-defined function in assembler, C, C++, COBOL, PL/I, or Java.

User-defined functions that are written in COBOL can include object-oriented extensions, just as other Db2 COBOL programs can. User-defined functions that are written in Java follow coding guidelines and restrictions specific to Java.

Your user-defined function can also access remote data by using DRDA access using CONNECT or SET CONNECTION statements.

Restrictions on user-defined function programs

Observe these restrictions when you write a user-defined function:

- Because Db2 uses the Resource Recovery Services attachment facility (RRSAF) as its interface with your user-defined function, you must not include RRSAF calls in your user-defined function. Db2 rejects any RRSAF calls that it finds in a user-defined function.
- If your user-defined function is not defined with parameters SCRATCHPAD or EXTERNAL ACTION, the user-defined function is not guaranteed to execute under the same task each time it is invoked.
- You cannot execute COMMIT or ROLLBACK statements in your user-defined function.
- You must close all cursors that were opened within a user-defined scalar function. Db2 returns an SQL error if a user-defined scalar function does not close all cursors that it opened before it completes.
- When you choose the language in which to write a user-defined function program, be aware of restrictions on the number of parameters that can be passed to a routine in that language. User-defined table functions in particular can require large numbers of parameters. Consult the programming guide for the language in which you plan to write the user-defined function for information about the number of parameters that can be passed.
- You cannot pass LOB file reference variables as parameters to user-defined functions.
- User-defined functions cannot return LOB file reference variables.
- You cannot pass parameters with the type XML to user-defined functions. You can specify tables or views that contain XML columns as table locator parameters. However, you cannot reference the XML columns in the body of the user-defined function.

Coding your user-defined function as a main program or as a subprogram

You can code your user-defined function as either a main program or a subprogram. The way that you code your program must agree with the way you defined the user-defined function: with the PROGRAM TYPE MAIN or PROGRAM TYPE SUB parameter. The main difference is that when a main program starts, Language Environment allocates the application program storage that the external user-defined function uses. When a main program ends, Language Environment closes files and releases dynamically allocated storage.

If you code your user-defined function as a subprogram and manage the storage and files yourself, you can get better performance. The user-defined function should always free any allocated storage before it exits. To keep data between invocations of the user-defined function, use a scratchpad.

You must code a user-defined table function that accesses external resources as a subprogram. Also ensure that the definer specifies the EXTERNAL ACTION parameter in the CREATE FUNCTION or ALTER FUNCTION statement. Program variables for a subprogram persist between invocations of the user-defined function, and use of the EXTERNAL ACTION parameter ensures that the user-defined function stays in the same address space from one invocation to another.

Parallelism considerations

If the definer specifies the parameter ALLOW PARALLEL in the definition of a user-defined scalar function, and the invoking SQL statement runs in parallel, the function can run under a parallel task. Db2 executes a separate instance of the user-defined function for each parallel task. When you write your function program, you need to understand how the following parameter values interact with ALLOW PARALLEL so that you can avoid unexpected results:

- SCRATCHPAD

  When an SQL statement invokes a user-defined function that is defined with the ALLOW PARALLEL parameter, Db2 allocates one scratchpad for each parallel task of each reference to the function. This can lead to unpredictable or incorrect results.

  For example, suppose that the user-defined function uses the scratchpad to count the number of times it is invoked. If a scratchpad is allocated for each parallel task, this count is the number of invocations done by the *parallel task* and not for the entire SQL statement, which is not the result that is wanted.
- FINAL CALL

If a user-defined function performs an external action, such as sending a note, for each final call to the function, one note is sent for each parallel task instead of once for the function invocation.

- EXTERNAL ACTION

  Some user-defined functions with external actions can receive incorrect results if the function is executed by parallel tasks.

  For example, if the function sends a note for each initial call to the function, one note is sent for each parallel task instead of once for the function invocation.

- NOT DETERMINISTIC

  A user-defined function that is non-deterministic can generate incorrect results if it is run under a parallel task.

  For example, suppose that you execute the following query under parallel tasks:

```
SELECT * FROM T1 WHERE C1 = COUNTER();
```

  COUNTER is a user-defined function that increments a variable in the scratchpad every time it is invoked. Counter is non-deterministic because the same input does not always produce the same output. Table T1 contains one column, C1, that has the following values:

```
1
2
3
4
5
6
7
8
9
10
```

  When the query is executed with no parallelism, Db2 invokes COUNTER once for each row of table T1, and there is one scratchpad for counter, which Db2 initializes the first time that COUNTER executes. COUNTER returns 1 the first time it executes, 2 the second time, and so on. The result table for the query has the following values:

```
1
2
3
4
5
6
7
8
9
10
```

  Now suppose that the query is run with parallelism, and Db2 creates three parallel tasks. Db2 executes the predicate WHERE C1 = COUNTER() for each parallel task. This means that each parallel task invokes its own instance of the user-defined function and has its own scratchpad. Db2 initializes the scratchpad to zero on the first call to the user-defined function for each parallel task.

  If parallel task 1 processes rows 1 to 3, parallel task 2 processes rows 4 to 6, and parallel task 3 processes rows 7 to 10, the following results occur:

  – When parallel task 1 executes, C1 has values 1, 2, and 3, and COUNTER returns values 1, 2, and 3, so the query returns values 1, 2, and 3.
  – When parallel task 2 executes, C1 has values 4, 5, and 6, but COUNTER returns values 1, 2, and 3, so the query returns no rows.
  – When parallel task 3, executes, C1 has values 7, 8, 9, and 10, but COUNTER returns values 1, 2, 3, and 4, so the query returns no rows.

  Thus, instead of returning the 10 rows that you might expect from the query, Db2 returns only 3 rows.

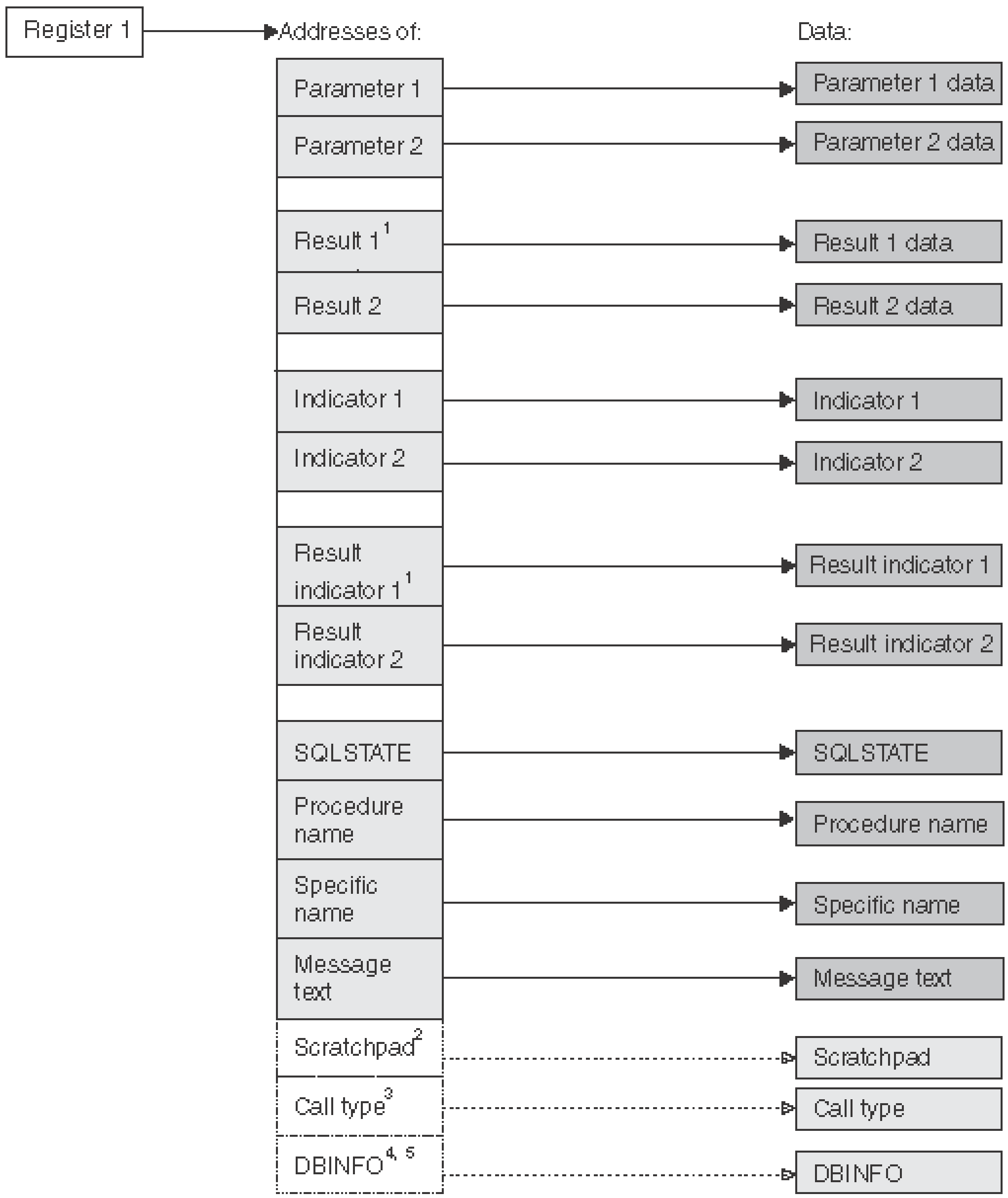
Related concepts

Java stored procedures and user-defined functions (Db2 Application Programming for Java)

Parameters for external user-defined functions

To receive parameters from and pass parameters to an invoker of an external user-defined function, you must understand the structure of the parameter list. You must also understand the meaning of each parameter, and whether Db2 or your user-defined function sets the value of each parameter.

The following figure shows the structure of the parameter list that Db2 passes to a user-defined function. An explanation of each parameter follows.



1. For a user-defined scalar function, only one result and one result indicator are passed.
2. Passed if the SCRATCHPAD option is specified in the user-defined function definition.
3. Passed if the FINAL CALL option is specified in a user-defined scalar function definition; always passed for a user-defined table function.
4. For PL/I, this value is the address of a pointer to the DBINFO data.
5. Passed if the DBINFO option is specified in the user-defined function definition.

Figure 8. Parameter conventions for a user-defined function

Input parameter values

Db2 obtains the input parameters from the invoker's parameter list, and your user-defined function receives those parameters according to the rules of the host language in which the user-defined function is written. The number of input parameters is the same as the number of parameters in the user-defined function invocation. If one of the parameters in the function invocation is an expression, Db2 evaluates the expression and assigns the result of the expression to the parameter.

For all data types except LOBs, ROWIDs, locators, and VARCHAR (with C language), see the tables listed in the following table for the host data types that are compatible with the data types in the user-defined function definition.

Table 43. Listing of tables of compatible data types

Language	Compatible data types table
Assembler	"Compatibility of SQL and language data types" on page 484
C	"Compatibility of SQL and language data types" on page 484
COBOL	"Compatibility of SQL and language data types" on page 484
PL/I	"Compatibility of SQL and language data types" on page 484

For LOBs, ROWIDs, and locators, see the following table for the assembler data types that are compatible with the data types in the user-defined function definition.

Table 44. Compatible assembler language declarations for LOBs, ROWIDs, and locators

SQL data type in definition	Assembler declaration
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`DS FL4`

BLOB(*n*)	If n <= 65535:
	`var     DS 0FL4`
	`var_length  DS FL4`
	`var_data DS CLn`
	If n > 65535:
	`var     DS 0FL4`
	`var_length  DS FL4`
	`var_data DS CL65535`
	`        ORG var_data+(n-65535)`

CLOB(*n*)	If n <= 65535:
	`var DS  0FL4`
	`var_length  DS FL4`
	`var_data DS CLn`
	If n > 65535:
	`var     DS 0FL4`
	`var_length  DS FL4`
	`var_data DS CL65535`
	`        ORG var_data+(n-65535)`

<table>
<caption>Table 44. Compatible assembler language declarations for LOBs, ROWIDs, and locators (continued)</caption>
<tr><th>SQL data type in definition</th><th>Assembler declaration</th></tr>
<tr><td>DBCLOB(n)</td><td>If n (=2*n) <= 65534:
<pre>var   DS 0FL4
var_length DS FL4
var_data DS CLm</pre>If n > 65534:
<pre>var   DS 0FL4
var_length DS FL4
var_data DS CL65534
        ORG var_data+(m-65534)</pre></td></tr>
<tr><td>ROWID</td><td><pre>DS HL2,CL40</pre></td></tr>
</table>

For LOBs, ROWIDs, VARCHARs, and locators see the following table for the C data types that are compatible with the data types in the user-defined function definition.

<table>
<caption>Table 45. Compatible C language declarations for LOBs, ROWIDs, VARCHARs, and locators</caption>
<tr><th>SQL data type in definition[1]</th><th>C declaration</th></tr>
<tr><td>TABLE LOCATOR
BLOB LOCATOR
CLOB LOCATOR
DBCLOB LOCATOR</td><td><pre>unsigned long</pre></td></tr>
<tr><td>BLOB(n)</td><td><pre>struct
{unsigned long length;
  char data[n];
}   var;</pre></td></tr>
<tr><td>CLOB(n)</td><td><pre>struct
{unsigned long length;
  char var_data[n];
}   var;</pre></td></tr>
<tr><td>DBCLOB(n)</td><td><pre>struct
{unsigned long length;
sqldbchar   data[n];
}   var;</pre></td></tr>
<tr><td>ROWID</td><td><pre>struct {
   short int length;
   char data[40];
} var;</pre></td></tr>
</table>

Table 45. Compatible C language declarations for LOBs, ROWIDs, VARCHARs, and locators (continued)

SQL data type in definition[1]	C declaration

VARCHAR(*n*)[2]	If PARAMETER VARCHAR NULTERM is specified or implied:
	`char data[n+1];`
	If PARAMETER VARCHAR STRUCTURE is specified:
	`struct {short len; char data[n]; } var;`

Note:

1. The SQLUDF file, which is in data set DSN1110.SDSNC.H, includes the typedef sqldbchar. Using sqldbchar lets you manipulate DBCS and Unicode UTF-16 data in the same format in which it is stored in Db2. sqldbchar also makes applications easier to port to other Db2 platforms.
2. This row does not apply to VARCHAR(*n*) FOR BIT DATA. BIT DATA is always passed in a structured representation.

For LOBs, ROWIDs, and locators, see the following table for the COBOL data types that are compatible with the data types in the user-defined function definition.

Table 46. Compatible COBOL declarations for LOBs, ROWIDs, and locators

SQL data type in definition	COBOL declaration
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`01 var  PIC S9(9) COMP-5`

BLOB(*n*)	`01 var.`
	`    49 var-LENGTH PIC S9(9) COMP-5.`
	`    49 var-DATA PIC X(n).`

CLOB(*n*)	`01 var.`
	`    49 var-LENGTH PIC S9(9) COMP-5.`
	`    49 var-DATA PIC X(n).`

DBCLOB(*n*)	`01 var.`
	`    49 var-LENGTH PIC S9(9) COMP-5.`
	`    49 var-DATA PIC G(n) DISPLAY-1.`

ROWID	`01 var.`
	`    49 var-LEN  PIC S9(4) COMP-5.`
	`    49 var-TEXT PIC X(40).`

For LOBs, ROWIDs, and locators, see the following table for the PL/I data types that are compatible with the data types in the user-defined function definition.

Table 47. Compatible PL/I declarations for LOBs, ROWIDs, and locators

SQL data type in definition	PL/I
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`BIN FIXED(31)`

BLOB(*n*)	If n <= 32767:
	`01 var,`
	`    03 var_LENGTH`
	`       BIN FIXED(31),`
	`    03 var_DATA`
	`       CHAR(n);`
	If n > 32767:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         CHAR(32767),`
	`      03 var_DATA2`
	`         CHAR(mod(n,32767));`

CLOB(*n*)	If n <= 32767:
	`01 var,`
	`    03 var_LENGTH`
	`       BIN FIXED(31),`
	`    03 var_DATA`
	`       CHAR(n);`
	If n > 32767:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         CHAR(32767),`
	`      03 var_DATA2`
	`         CHAR(mod(n,32767));`

DBCLOB(*n*)	If n <= 16383:
	`01 var,`
	`   03 var_LENGTH`
	`      BIN FIXED(31),`
	`   03 var_DATA`
	`      GRAPHIC(n);`
	If n > 16383:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         GRAPHIC(16383),`
	`      03 var_DATA2`
	`         GRAPHIC(mod(n,16383));`

ROWID	`CHAR(40) VAR;`

Result parameters: Set these values in your user-defined function before exiting. For a user-defined scalar function, you return one result parameter. For a user-defined table function, you return the same number of parameters as columns in the RETURNS TABLE clause of the CREATE FUNCTION statement. Db2 allocates a buffer for each result parameter value and passes the buffer address to the user-defined function. Your user-defined function places each result parameter value in its buffer. You must ensure that the length of the value you place in each output buffer does not exceed the buffer length. Use the SQL data type and length in the CREATE FUNCTION statement to determine the buffer length.

See “Parameters for external user-defined functions” on page 187 to determine the host data type to use for each result parameter value. If the CREATE FUNCTION statement contains a CAST FROM clause, use a data type that corresponds to the SQL data type in the CAST FROM clause. Otherwise, use a data type that corresponds to the SQL data type in the RETURNS or RETURNS TABLE clause.

To improve performance for user-defined table functions that return many columns, you can pass values for a subset of columns to the invoker. For example, a user-defined table function might be defined to return 100 columns, but the invoker needs values for only two columns. Use the DBINFO parameter to indicate to Db2 the columns for which you will return values. Then return values for only those columns. See DBINFO for information about how to indicate the columns of interest.

Input parameter indicators: These are SMALLINT values, which Db2 sets before it passes control to the user-defined function. You use the indicators to determine whether the corresponding input parameters are null. The number and order of the indicators are the same as the number and order of the input parameters. On entry to the user-defined function, each indicator contains one of these values:

0
: The input parameter value is not null.

negative
: The input parameter value is null.

Code the user-defined function to check all indicators for null values unless the user-defined function is defined with RETURNS NULL ON NULL INPUT. A user-defined function defined with RETURNS NULL ON NULL INPUT executes only if all input parameters are not null.

Result indicators: These are SMALLINT values, which you must set before the user-defined function ends to indicate to the invoking program whether each result parameter value is null. A user-defined scalar function has one result indicator. A user-defined table function has the same number of result indicators as the number of result parameters. The order of the result indicators is the same as the order of the result parameters. Set each result indicator to one of these values:

0 or positive
: The result parameter is not null.

negative
: The result parameter is null.

SQLSTATE value: This CHAR(5) value represents the SQLSTATE that is passed in to the program from the database manager. The initial value is set to ‘00000'. Although the SQLSTATE is usually not set by the program, it can be set as the result SQLSTATE that is used to return an error or a warning. Returned values that start with anything other than ‘00', ‘01', or ‘02' are error conditions.

User-defined function name: Db2 sets this value in the parameter list before the user-defined function executes. This value is VARCHAR(257): 128 bytes for the schema name, 1 byte for a period, and 128 bytes for the user-defined function name. If you use the same code to implement multiple versions of a user-defined function, you can use this parameter to determine which version of the function the invoker wants to execute.

Specific name: Db2 sets this value in the parameter list before the user-defined function executes. This value is VARCHAR(128) and is either the specific name from the CREATE FUNCTION statement or a specific name that Db2 generated. If you use the same code to implement multiple versions of a user-defined function, you can use this parameter to determine which version of the function the invoker wants to execute.

Diagnostic message: Your user-defined function can set this CHAR or VARCHAR value to a character string of up to 1000 bytes before exiting. Use this area to pass descriptive information about an error or warning to the invoker.

Db2 allocates a buffer for this area and passes you the buffer address in the parameter list. At least the first 17 bytes of the value you put in the buffer appear in the SQLERRMC field of the SQLCA that is returned to the invoker. The exact number of bytes depends on the number of other tokens in SQLERRMC. Do not use X'FF' in your diagnostic message. Db2 uses this value to delimit tokens.

Scratchpad: If the definer specified SCRATCHPAD in the CREATE FUNCTION statement, Db2 allocates a buffer for the scratchpad area and passes its address to the user-defined function. Before the user-defined function is invoked for the first time in an SQL statement, Db2 sets the length of the scratchpad in the first 4 bytes of the buffer and then sets the scratchpad area to X'00'. Db2 does not reinitialize the scratchpad between invocations of a correlated subquery.

You must ensure that your user-defined function does not write more bytes to the scratchpad than the scratchpad length.

Call type: For a user-defined scalar function, if the definer specified FINAL CALL in the CREATE FUNCTION statement, Db2 passes this parameter to the user-defined function. For a user-defined table function, Db2 always passes this parameter to the user-defined function.

On entry to a *user-defined scalar function,* the call type parameter has one of the following values:

-1
: This is the *first call* to the user-defined function for the SQL statement. For a first call, all input parameters are passed to the user-defined function. In addition, the scratchpad, if allocated, is set to binary zeros.

0
: This is a *normal call*. For a normal call, all the input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

1
: This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

  This type of final call occurs when the invoking application explicitly closes a cursor. When a value of 1 is passed to a user-defined function, the user-defined function can execute SQL statements.

255
: This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

  This type of final call occurs when the invoking application executes a COMMIT or ROLLBACK statement, or when the invoking application abnormally terminates. When a value of 255 is passed to the user-defined function, the user-defined function cannot execute any SQL statements, except for CLOSE CURSOR. If the user-defined function executes any close cursor statements during this type of final call, the user-defined function should tolerate SQLCODE -501 because Db2 might have already closed cursors before the final call.

During the first call, your user-defined scalar function should acquire any system resources it needs. During the final call, the user-defined scalar function should release any resources it acquired during the first call. The user-defined scalar function should return a result value only during normal calls. Db2 ignores any results that are returned during a final call. However, the user-defined scalar function can set the SQLSTATE and diagnostic message area during the final call.

If an invoking SQL statement contains more than one user-defined scalar function, and one of those user-defined functions returns an error SQLSTATE, Db2 invokes all of the user-defined functions for a final call, and the invoking SQL statement receives the SQLSTATE of the first user-defined function with an error.

On entry to a *user-defined table function,* the call type parameter has one of the following values:

-2
: This is the *first call* to the user-defined function for the SQL statement. A first call occurs only if the FINAL CALL keyword is specified in the user-defined function definition. For a first call, all input parameters are passed to the user-defined function. In addition, the scratchpad, if allocated, is set to binary zeros.

-1
: This is the *open call* to the user-defined function by an SQL statement. If FINAL CALL is not specified in the user-defined function definition, all input parameters are passed to the user-defined function, and the scratchpad, if allocated, is set to binary zeros during the open call. If FINAL CALL is specified for the user-defined function, Db2 does not modify the scratchpad.

0
: This is a *fetch call* to the user-defined function by an SQL statement. For a fetch call, all input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

1
: This is a *close call*. For a close call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

2
: This is a *final call*. This type of final call occurs only if FINAL CALL is specified in the user-defined function definition. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

  This type of final call occurs when the invoking application executes a CLOSE CURSOR statement.

255
: This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, Db2 does not modify it.

  This type of final call occurs when the invoking application executes a COMMIT or ROLLBACK statement, or when the invoking application abnormally terminates. When a value of 255 is passed to the user-defined function, the user-defined function cannot execute any SQL statements, except for CLOSE CURSOR. If the user-defined function executes any close cursor statements during this type of final call, the user-defined function should tolerate SQLCODE -501 because Db2 might have already closed cursors before the final call.

If a user-defined table function is defined with FINAL CALL, the user-defined function should allocate any resources it needs during the first call and release those resources during the final call that sets a value of 2.

If a user-defined table function is defined with NO FINAL CALL, the user-defined function should allocate any resources it needs during the open call and release those resources during the close call.

During a fetch call, the user-defined table function should return a row. If the user-defined function has no more rows to return, it should set the SQLSTATE to 02000.

During the close call, a user-defined table function can set the SQLSTATE and diagnostic message area.

If a user-defined table function is invoked from a subquery, the user-defined table function receives a CLOSE call for each invocation of the subquery within the higher level query, and a subsequent OPEN call for the next invocation of the subquery within the higher level query.

DBINFO: If the definer specified DBINFO in the CREATE FUNCTION statement, Db2 passes the DBINFO structure to the user-defined function. DBINFO contains information about the environment of the user-defined function caller. It contains the following fields, in the order shown:

Location name length
: An unsigned 2-byte integer field. It contains the length of the location name in the next field.

Location name
: A 128-byte character field. It contains the name of the location to which the invoker is currently connected.

Authorization ID length
An unsigned 2-byte integer field. It contains the length of the authorization ID in the next field.

Authorization ID
A 128-byte character field. It contains the authorization ID of the application from which the user-defined function is invoked, padded on the right with blanks. If this user-defined function is nested within other user-defined functions, this value is the authorization ID of the application that invoked the highest-level user-defined function.

Subsystem code page
A 48-byte structure that consists of 10 integer fields and an eight-byte reserved area. These fields provide information about the CCSIDs of the subsystem from which the user-defined function is invoked.

Table qualifier length
An unsigned 2-byte integer field. It contains the length of the table qualifier in the next field. If the table name field is not used, this field contains 0.

Table qualifier
A 128-byte character field. It contains the qualifier of the table that is specified in the table name field.

Table name length
An unsigned 2-byte integer field. It contains the length of the table name in the next field. If the table name field is not used, this field contains 0.

Table name
A 128-byte character field. This field contains the name of the table for the update or insert operation if the reference to the user-defined function in the invoking SQL statement is in one of the following places:

- The right side of a SET clause in an update operation
- In the VALUES list of an insert operation

Otherwise, this field is blank.

Column name length
An unsigned 2-byte integer field. It contains the length of the column name in the next field. If no column name is passed to the user-defined function, this field contains 0.

Column name
A 128-byte character field. This field contains the name of the column that the update or insert operation modifies if the reference to the user-defined function in the invoking SQL statement is in one of the following places:

- The right side of a SET clause in an update operation
- In the VALUES list of an insert operation

Otherwise, this field is blank.

Product information
An 8-byte character field that identifies the product on which the user-defined function executes.

The format of product identifier values is *pppvvrrm*, where *ppp* is a 3-letter product code (such as DSN for Db2), *vv* is the version, *rr* is the release, and *m* is the modification level. For example, DSN11015 identifies Db2 11 in new-function mode, the value is ‘DSN11015'. The product code (*ppp*) is one of the following values:

AQT for IBM Db2 Analytics Accelerator for z/OS
ARI for DB2 Server for VSE & VM
DSN for Db2 for z/OS
HTP for non-secure HTTP URL connections for Db2 native REST services
HTS for secure HTTPS connections for Db2 native REST services
JCC for IBM Data Server Driver for JDBC and SQLJ
QSQ for DB2 for i

SQL for Db2 for Linux®, UNIX, and Windows

Modification (*m*) values have the following meanings:

0
Conversion and enabling-new-function modes for migration from DB2 10 (CM10, CM10*, ENFM10, and ENFM10*)

5
New-function mode.

Reserved area
2 bytes.

Operating system
A 4-byte integer field. It identifies the operating system on which the program that invokes the user-defined function runs. The value is one of these:

0
Unknown

1
OS/2

3
Windows

4
AIX®

5
Windows NT

6
HP-UX

7
Solaris

8
z/OS

13
Siemens Nixdorf

15
Windows 95

16
SCO UNIX

18
Linux

19
DYNIX/ptx®

24
Linux for S/390®

25
Linux on IBM Z

26
Linux/IA64

27
Linux/PPC

28
Linux/PPC64

29
Linux/AMD64

400®
iSeries

Number of entries in table function column list
An unsigned 2-byte integer field.

Reserved area
26 bytes.

Table function column list pointer
If a table function is defined, this field is a pointer to an array that contains 1000 2-byte integers. Db2 dynamically allocates the array. If a table function is not defined, this pointer is null.

Only the first *n* entries, where *n* is the value in the field entitled number of entries in table function column list, are of interest. *n* is greater than or equal to 0 and less than or equal to the number result columns defined for the user-defined function in the RETURNS TABLE clause of the CREATE FUNCTION statement. The values correspond to the numbers of the columns that the invoking statement needs from the table function. A value of 1 means the first defined result column, 2 means the second defined result column, and so on. The values can be in any order. If *n* is equal to 0, the first array element is 0. This is the case for a statement like the following one, where the invoking statement needs no column values.

```
SELECT COUNT(*) FROM TABLE(TF(…)) AS QQ
```

This array represents an opportunity for optimization. The user-defined function does not need to return all values for all the result columns of the table function. Instead, the user-defined function can return only those columns that are needed in the particular context, which you identify by number in the array. However, if this optimization complicates the user-defined function logic enough to cancel the performance benefit, you might choose to return every defined column.

Unique application identifier
This field is a pointer to a string that uniquely identifies the application's connection to Db2. The string is regenerated for each connection to Db2.

The string is the LUWID, which consists of a fully-qualified LU network name followed by a period and an LUW instance number. The LU network name consists of a 1- to 8-character network ID, a period, and a 1- to 8-character network LU name. The LUW instance number consists of 12 hexadecimal characters that uniquely identify the unit of work.

Reserved area
20 bytes.

If you write your user-defined function in C or C++, you can use the declarations in member SQLUDF of DSN1110.SDSNC.H for many of the passed parameters. To include SQLUDF, make these changes to your program:

- Put this statement in your source code:

```
#include <sqludf.h>
```

- Include the DSN1110.SDSNC.H data set in the SYSLIB concatenation for the compiler step of your program preparation job.
- Specify the NOMARGINS and NOSEQUENCE options in the compiler step of your program preparation job.

Examples of receiving parameters in a user-defined function:

The following examples show how a user-defined function that is written in each of the supported host languages receives the parameter list that is passed by Db2.

These examples assume that the user-defined function is defined with the SCRATCHPAD, FINAL CALL, and DBINFO parameters.

Assembler: The follow figure shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result. For an assembler language user-defined function that is a subprogram, the conventions are the same. In either case, you must include the CEEENTRY and CEEEXIT macros.

```
MYMAIN   CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
         USING PROGAREA,R13

         L     R7,0(R1)            GET POINTER TO PARM1
         MVC   PARM1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM1
         L     R7,4(R1)            GET POINTER TO PARM2
         MVC   PARM2(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM2
         L     R7,12(R1)           GET POINTER TO INDICATOR 1
         MVC   F_IND1(2),0(R7)  MOVE PARM1 INDICATOR TO LOCAL STORAGE
         LH    R7,F_IND1        MOVE PARM1 INDICATOR INTO R7
         LTR   R7,R7               CHECK IF IT IS NEGATIVE
         BM    NULLIN              IF SO, PARM1 IS NULL
         L     R7,16(R1)           GET POINTER TO INDICATOR 2
         MVC   F_IND2(2),0(R7)  MOVE PARM2 INDICATOR TO LOCAL STORAGE
         LH    R7,F_IND2        MOVE PARM2 INDICATOR INTO R7
         LTR   R7,R7               CHECK IF IT IS NEGATIVE
         BM    NULLIN              IF SO, PARM2 IS NULL
         ⋮
NULLIN   L     R7,8(R1)            GET ADDRESS OF AREA FOR RESULT
         MVC   0(9,R7),RESULT      MOVE A VALUE INTO RESULT AREA
         L     R7,20(R1)           GET ADDRESS OF AREA FOR RESULT IND
         MVC   0(2,R7),=H'0'       MOVE A VALUE INTO INDICATOR AREA
         ⋮
         CEETERM  RC=0
*******************************************************************
*  VARIABLE DECLARATIONS AND EQUATES                              *
*******************************************************************
R1       EQU   1                   REGISTER 1
R7       EQU   7                   REGISTER 7
PPA      CEEPPA  ,                 CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,                   PLACE LITERAL POOL HERE
PROGAREA DSECT
         ORG   *+CEEDSASZ          LEAVE SPACE FOR DSA FIXED PART
PARM1    DS    F                   PARAMETER 1
PARM2    DS    F                   PARAMETER 2
RESULT   DS    CL9                 RESULT
F_IND1   DS    H                   INDICATOR FOR PARAMETER 1
F_IND2   DS    H                   INDICATOR FOR PARAMETER 2
F_INDR   DS    H                   INDICATOR FOR RESULT

PROGSIZE EQU   *-PROGAREA
         CEEDSA  ,                 MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                 MAPPING OF THE COMMON ANCHOR AREA
         END   MYMAIN
```

C or C++: For C or C++ user-defined functions, the conventions for passing parameters are different for main programs and subprograms.

For subprograms, you pass the parameters directly. For main programs, you use the standard argc and argv variables to access the input and output parameters:

- The argv variable contains an array of pointers to the parameters that are passed to the user-defined function. All string parameters that are passed back to Db2 must be null terminated.
  - argv[0] contains the address of the load module name for the user-defined function.
  - argv[1] through argv[*n*] contain the addresses of parameters 1 through *n*.
- The argc variable contains the number of parameters that are passed to the external user-defined function, including argv[0].

The following figure shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result.

```
#include <stdlib.h>
#include <stdio.h>

main(argc,argv)
  int argc;
  char *argv[];
 {
```

```
/*************************************************/
/* Assume that the user-defined function invocation*/
/* included 2 input  parameters in the parameter   */
/* list.  Also assume that the definition includes */
/* the SCRATCHPAD, FINAL CALL, and DBINFO options, */
/* so DB2 passes the scratchpad, calltype, and     */
/* dbinfo parameters.                              */
/* The argv vector contains these entries:         */
/*      argv[0]         1    load module name      */
/*      argv[1-2]       2    input parms           */
/*      argv[3]         1    result parm           */
/*      argv[4-5]       2    null indicators       */
/*      argv[6]         1    result null indicator */
/*      argv[7]         1    SQLSTATE variable     */
/*      argv[8]         1    qualified func name   */
/*      argv[9]         1    specific func name    */
/*      argv[10]        1    diagnostic string     */
/*      argv[11]        1    scratchpad            */
/*      argv[12]        1    call type             */
/*      argv[13]      + 1    dbinfo                */
/*                  ------                         */
/*                    14     for the argc variable */
/*************************************************/
if argc<>14
{
⋮
  /**********************************************************/
  /* This section would contain the code executed if the    */
  /* user-defined function is invoked with the wrong number */
  /* of parameters.                                         */
  /**********************************************************/
}
```

```
/*************************************************/
/* Assume the first parameter is an integer.       */
/* The following code shows how to copy the integer*/
/* parameter into the application storage.         */
/*************************************************/
int parm1;
parm1 = *(int *) argv[1];

/*************************************************/
/* Access the null indicator for the first         */
/* parameter on the invoked user-defined function  */
/* as follows:                                     */
/*************************************************/
short int ind1;
ind1 = *(short int *) argv[4];

/*************************************************/
/* Use the following expression to assign          */
/* 'xxxxx' to the SQLSTATE returned to caller on   */
/* the SQL statement that contains the invoked     */
/* user-defined function.                          */
/*************************************************/
strcpy(argv[7],"xxxxx");

/*************************************************/
/* Obtain the value of the qualified function      */
/* name with this expression.                      */
/*************************************************/
char f_func[28];
strcpy(f_func,argv[8]);
/*************************************************/
/* Obtain the value of the specific function       */
/* name with this expression.                      */
/*************************************************/
char f_spec[19];
strcpy(f_spec,argv[9]);

/*************************************************/
/* Use the following expression to assign          */
/* 'yyyyyyyy' to the diagnostic string returned    */
/* in the SQLCA associated with the invoked        */
/* user-defined function.                          */
/*************************************************/
strcpy(argv[10],"yyyyyyyy");

/*************************************************/
/* Use the following expression to assign the      */
/* result of the function.                         */
```

```
   /*************************************************/
   char l_result[11];
   strcpy(argv[3],l_result);

   ⋮
 }
```

The following figure shows the parameter conventions for a user-defined scalar function written as a C subprogram that receives two parameters and returns one result.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sqludf.h>

void myfunc(long *parm1, char parm2[11], char result[11],
            short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_msgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
{
    /*************************************************/
    /* Declare local copies of parameters             */
    /*************************************************/
    int l_p1;
    char l_p2[11];
    short int l_ind1;
    short int l_ind2;
    char ludf_sqlstate[6];     /* SQLSTATE               */
    char ludf_fname[138];       /* function name          */
    char ludf_specname[129];   /* specific function name */
    char ludf_msgtext[71]      /* diagnostic message text*/
    sqludf_scratchpad *ludf_scratchpad; /* scratchpad    */
    long *ludf_call_type;      /* call type              */
    sqludf_dbinfo *ludf_dbinfo /* dbinfo                 */
    /*************************************************/
    /* Copy each of the parameters in the parameter    */
    /* list into a local variable to demonstrate       */
    /* how the parameters can be referenced.           */
    /*************************************************/

    l_p1 = *parm1;
    strcpy(l_p2,parm2);
    l_ind1 = *f_ind1;
    l_ind1 = *f_ind2;
    strcpy(ludf_sqlstate,udf_sqlstate);
    strcpy(ludf_fname,udf_fname);
    strcpy(ludf_specname,udf_specname);
    l_udf_call_type = *udf_call_type;
    strcpy(ludf_msgtext,udf_msgtext);
    memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
    memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));
    ⋮
 }
```

The following figure shows the parameter conventions for a user-defined scalar function that is written as a C++ subprogram that receives two parameters and returns one result. This example demonstrates that you must use an extern "C" modifier to indicate that you want the C++ subprogram to receive parameters according to the C linkage convention. This modifier is necessary because the CEEPIPI CALL_SUB interface, which Db2 uses to call the user-defined function, passes parameters using the C linkage convention.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
#include <sqludf.h>

extern "C" void myfunc(long *parm1, char parm2[11],
            char result[11], short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_msgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
```

```
{
    /***************************************************/
    /* Define local copies of parameters.              */
    /***************************************************/
    int l_p1;
    char l_p2[11];
    short int l_ind1;
    short int l_ind2;
    char ludf_sqlstate[6];     /* SQLSTATE               */
    char ludf_fname[138];      /* function name          */
    char ludf_specname[129];   /* specific function name */
    char ludf_msgtext[71]      /* diagnostic message text*/
    sqludf_scratchpad *ludf_scratchpad; /* scratchpad          */
    long *ludf_call_type;      /* call type                    */
    sqludf_dbinfo *ludf_dbinfo /* dbinfo                       */
    /***************************************************/
    /* Copy each of the parameters in the parameter    */
    /* list into a local variable to demonstrate       */
    /* how the parameters can be referenced.           */
    /***************************************************/
    l_p1 = *parm1;
    strcpy(l_p2,parm2);
    l_ind1 = *f_ind1;
    l_ind1 = *f_ind2;
    strcpy(ludf_sqlstate,udf_sqlstate);
    strcpy(ludf_fname,udf_fname);
    strcpy(ludf_specname,udf_specname);
    l_udf_call_type = *udf_call_type;
    strcpy(ludf_msgtext,udf_msgtext);
    memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
    memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));
    ⋮
 }
```

COBOL: The following figure shows the parameter conventions for a user-defined table function that is written as a main program that receives two parameters and returns two results. For a COBOL user-defined function that is a subprogram, the conventions are the same.

```
 CBL APOST,RES,RENT
       IDENTIFICATION DIVISION.
       ⋮
       DATA DIVISION.
       ⋮
       LINKAGE SECTION.
      *********************************************************
      * Declare each of the parameters                        *
      *********************************************************
       01  UDFPARM1 PIC S9(9) USAGE COMP.
       01  UDFPARM2 PIC X(10).
      ⋮
      *********************************************************
      * Declare these variables for result parameters         *
      *********************************************************
       01  UDFRESULT1 PIC X(10).
       01  UDFRESULT2 PIC X(10).
      ⋮
      *********************************************************
      * Declare a null indicator for each parameter           *
      *********************************************************
       01  UDF-IND1 PIC S9(4) USAGE COMP.
       01  UDF-IND2 PIC S9(4) USAGE COMP.
      ⋮
      *********************************************************
      * Declare a null indicator for result parameter         *
      *********************************************************
       01  UDF-RIND1 PIC S9(4) USAGE COMP.
       01  UDF-RIND2 PIC S9(4) USAGE COMP.
      ⋮
      *********************************************************
      * Declare the SQLSTATE that can be set by the           *
      * user-defined function                                 *
      *********************************************************
       01  UDF-SQLSTATE PIC X(5).
      *********************************************************
      * Declare the qualified function name                   *
      *********************************************************
       01  UDF-FUNC.
           49 UDF-FUNC-LEN PIC 9(4) USAGE BINARY.
           49 UDF-FUNC-TEXT PIC X(137).
```

```
*********************************************************
* Declare the specific function name                    *
*********************************************************
 01  UDF-SPEC.
     49 UDF-SPEC-LEN PIC 9(4) USAGE BINARY.
     49 UDF-SPEC-TEXT PIC X(128).
*********************************************************
* Declare SQL diagnostic message token                  *
*********************************************************
 01  UDF-DIAG.
     49 UDF-DIAG-LEN PIC 9(4) USAGE BINARY.
     49 UDF-DIAG-TEXT PIC X(1000).
```

```
*********************************************************
* Declare the scratchpad                                *
*********************************************************
 01  UDF-SCRATCHPAD.
     49 UDF-SPAD-LEN PIC 9(9) USAGE BINARY.
     49 UDF-SPAD-TEXT PIC X(100).
*********************************************************
* Declare the call type                                 *
*********************************************************
 01  UDF-CALL-TYPE PIC 9(9) USAGE BINARY.
*********************************************************
* CONSTANTS FOR DB2-EBCODING-SCHEME.                    *
*********************************************************
 77 SQLUDF-ASCII PIC 9(9) VALUE 1.
 77 SQLUDF-EBCDIC PIC 9(9) VALUE 2.
 77 SQLUDF-UNICODE PIC 9(9) VALUE 3.
*********************************************************
* Structure used for DBINFO                             *
*********************************************************
 01  SQLUDF-DBINFO.
*       location name length
     05 DBNAMELEN PIC 9(4) USAGE BINARY.
*       location name
     05 DBNAME PIC X(128).
*       authorization ID length
     05 AUTHIDLEN PIC 9(4) USAGE BINARY.
*       authorization ID
     05 AUTHID PIC X(128).
*       environment CCSID information
     05  CODEPG PIC X(48).
     05 CDPG-DB2 REDEFINES CODEPG.
        10 DB2-CCSIDS OCCURS 3 TIMES.
           15 DB2-SBCS   PIC 9(9) USAGE BINARY.
           15 DB2-DBCS   PIC 9(9) USAGE BINARY.
           15 DB2-MIXED  PIC 9(9) USAGE BINARY.
        10 ENCODING-SCHEME   PIC 9(9) USAGE BINARY.
        10 RESERVED    PIC X(8).
* other platform-specific deprecated CCSID structures not included here
*       schema name length
     05 TBSCHEMALEN PIC 9(4) USAGE BINARY.
*       schema name
     05 TBSCHEMA PIC X(128).
*       table name length
     05 TBNAMELEN PIC 9(4) USAGE BINARY.
*       table name
     05 TBNAME PIC X(128).
*       column name length
     05 COLNAMELEN PIC 9(4) USAGE BINARY.
*       column name
     05 COLNAME PIC X(128).
*       product information
     05 VER-REL PIC X(8).
*       reserved for expansion
     05 RESD0 PIC X(2).
*       platform type
     05 PLATFORM PIC 9(9) USAGE BINARY.
*       number of entries in tfcolumn list array (tfcolumn, below)
     05 NUMTFCOL PIC 9(4) USAGE BINARY.
```

```
*       reserved for expansion
     05 RESD1 PIC X(26).
*       tfcolumn will be allocated dynamically if TF is defined
*       otherwise this will be a null pointer
     05 TFCOLUMN USAGE IS POINTER.
*       Application identifier
```

```
           05 APPL-ID USAGE IS POINTER.
      *       reserved for expansion
           05 RESD2 PIC X(20).
      *
       PROCEDURE DIVISION USING UDFPARM1, UDFPARM2, UDFRESULT1,
                              UDFRESULT2, UDF-IND1, UDF-IND2,
                              UDF-RIND1, UDF-RIND2,
                              UDF-SQLSTATE, UDF-FUNC, UDF-SPEC,
                              UDF-DIAG, UDF-SCRATCHPAD,
                              UDF-CALL-TYPE, SQLUDF-DBINFO.
```

PL/I: The following figure shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result. For a PL/I user-defined function that is a subprogram, the conventions are the same.

```
*PROCESS SYSTEM(MVS);
 MYMAIN: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
              UDF_IND1, UDF_IND2, UDF_INDR,
              UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
              UDF_DIAG_MSG, UDF_SCRATCHPAD,
              UDF_CALL_TYPE, UDF_DBINFO)
         OPTIONS(MAIN NOEXECOPS REENTRANT);

 DCL UDF_PARM1 BIN FIXED(31);    /* first parameter              */
 DCL UDF_PARM2 CHAR(10);         /* second parameter             */
 DCL UDF_RESULT CHAR(10);        /* result parameter             */
 DCL UDF_IND1 BIN FIXED(15);     /* indicator for 1st parm       */
 DCL UDF_IND2 BIN FIXED(15);     /* indicator for 2nd parm       */
 DCL UDF_INDR BIN FIXED(15);     /* indicator for result         */
 DCL UDF_SQLSTATE CHAR(5);       /* SQLSTATE returned to DB2     */
 DCL UDF_NAME CHAR(137) VARYING; /* Qualified function name      */
 DCL UDF_SPEC_NAME CHAR(128) VARYING; /* Specific function name  */
 DCL UDF_DIAG_MSG CHAR(70) VARYING; /* Diagnostic string         */
 DCL 01 UDF_SCRATCHPAD            /* Scratchpad                  */
      03 UDF_SPAD_LEN BIN FIXED(31),
      03 UDF_SPAD_TEXT CHAR(100);
 DCL UDF_CALL_TYPE BIN FIXED(31); /* Call Type                  */
 DCL DBINFO PTR;
    /* CONSTANTS FOR DB2_ENCODING_SCHEME */
 DCL SQLUDF_ASCII BIN FIXED(15) INIT(1);
 DCL SQLUDF_EBCDIC BIN FIXED(15) INIT(2);
 DCL SQLUDF_MIXED BIN FIXED(15) INIT(3);
```

```
   DCL 01 UDF_DBINFO BASED(DBINFO),              /* Dbinfo                 */
          03 UDF_DBINFO_LLEN BIN FIXED(15),      /* location length        */
          03 UDF_DBINFO_LOC  CHAR(128),          /* location name          */
          03 UDF_DBINFO_ALEN BIN FIXED(15),      /* auth ID  length        */
          03 UDF_DBINFO_AUTH CHAR(128),          /* authorization ID       */
          03 UDF_DBINFO_CDPG,                    /* environment CCSID info */
            05 DB2_CCSIDS(3),
              07 R1               BIN FIXED(15), /* Reserved               */
              07 DB2_SBCS  BIN FIXED(15),        /* SBCS CCSID             */
              07 R2               BIN FIXED(15), /* Reserved               */
              07 DB2_DBCS  BIN FIXED(15),        /* DBCS CCSID             */
              07 R3               BIN FIXED(15), /* Reserved               */
              07 DB2_MIXED  BIN FIXED(15),       /* MIXED CCSID            */
            05 DB2_ENCODING_SCHEME BIN FIXED(31),
            05 DB2_CCSID_RESERVED CHAR(8),
          03 UDF_DBINFO_SLEN BIN FIXED(15),       /* schema length         */
          03 UDF_DBINFO_SCHEMA  CHAR(128),        /* schema name           */
          03 UDF_DBINFO_TLEN BIN FIXED(15),       /* table length          */
          03 UDF_DBINFO_TABLE   CHAR(128),        /* table name            */
          03 UDF_DBINFO_CLEN BIN FIXED(15),       /* column length         */
          03 UDF_DBINFO_COLUMN  CHAR(128),        /* column name           */
          03 UDF_DBINFO_RELVER  CHAR(8),          /* DB2 release level     */
          03  UDF_DBINFO_RESERV0 CHAR(2),          /* reserved              */
          03 UDF_DBINFO_PLATFORM BIN FIXED(31),   /* database platform     */
          03 UDF_DBINFO_NUMTFCOL BIN FIXED(15),   /* # of TF columns used  */
          03 UDF_DBINFO_RESERV1 CHAR(26),         /* reserved              */
          03 UDF_DBINFO_TFCOLUMN PTR,             /* -> TFcolumn list      */
          03 UDF_DBINFO_APPLID   PTR,             /* -> application id     */
          03 UDF_DBINFO_RESERV2 CHAR(20);         /* reserved              */
 ⋮
```

Related reference

CREATE FUNCTION (external scalar) (Db2 SQL)

Making a user-defined function reentrant

A reentrant user-defined function is a function for which a single copy of the function can be used concurrently by two or more processes.

Procedure

Compiling and link-editing your user-defined function as reentrant is recommended. (For an assembler program, you must also code the user-defined function to be reentrant.)

Reentrant user-defined functions have the following advantages:

- The operating system does not need to load the user-defined function into storage every time the user-defined function is called.
- Multiple tasks in a WLM-established stored procedures address space can share a single copy of the user-defined function. This decreases the amount of virtual storage that is needed for code in the address space.

If your user-defined function consists of several programs, you must bind each program that contains SQL statements into a separate package. The definer of the user-defined function must have EXECUTE authority for all packages that are part of the user-defined function.

When the primary program of a user-defined function calls another program, Db2 uses the CURRENT PACKAGE PATH special register to determine the list of collections to search for the called program's package. The primary program can change this collection ID by executing the statement SET CURRENT PACKAGE PATH.

If the value of CURRENT PACKAGE PATH is blank or an empty string, Db2 uses the CURRENT PACKAGESET special register to determine the collection to search for the called program's package. The primary program can change this value by executing the statement SET CURRENT PACKAGESET.

If both special registers CURRENT PACKAGE PATH and CURRENT PACKAGESET contain a blank value, Db2 uses the method described in "Binding an application plan" on page 857 to search for the package.

Special registers in a user-defined function or a stored procedure

You can use all special registers in a user-defined function or a stored procedure. However, you can modify only some of those special registers.

After a user-defined function or a stored procedure completes, Db2 restores all special registers to the values they had before invocation.

The following table shows information that you need when you use special registers in a user-defined function or stored procedure.

Table 48. Characteristics of special registers in a user-defined function or a stored procedure

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?
CURRENT ACCELERATOR	Inherited from the invoking application[6]; otherwise, no preferred accelerator is used and Db2 will determine the target accelerator	The ACCELERATOR bind option value if specified for the user-defined function or stored procedure package; otherwise, no preferred accelerator is used and Db2 will determine the target accelerator	Yes

Table 48. Characteristics of special registers in a user-defined function or a stored procedure (continued)

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?
CURRENT APPLICATION COMPATIBILITY	The value of bind option APPLCOMPAT for the user-defined function or stored procedure package	The value of bind option APPLCOMPAT for the user-defined function or stored procedure package	Yes
CURRENT APPLICATION ENCODING SCHEME	The value of bind option ENCODING for the user-defined function or stored procedure package	The value of bind option ENCODING for the user-defined function or stored procedure package	Yes
CURRENT CLIENT_ACCTNG	Inherited from the invoking application	Inherited from the invoking application	Not applicable[5]
CURRENT CLIENT_APPLNAME	Inherited from the invoking application	Inherited from the invoking application	Not applicable[5]
CURRENT CLIENT_USERID	Inherited from the invoking application	Inherited from the invoking application	Not applicable[5]
CURRENT CLIENT_WRKSTNNAME	Inherited from the invoking application	Inherited from the invoking application	Not applicable[5]
CURRENT DATE	New value for each SQL statement in the user-defined function or stored procedure package[1]	New value for each SQL statement in the user-defined function or stored procedure package[1]	Not applicable[5]
CURRENT DEBUG MODE	Inherited from the invoking application	DISALLOW	Yes
CURRENT DECFLOAT ROUNDING MODE	Inherited from the invoking application	The value of bind option ROUNDING for the user-defined function or stored procedure package	Yes
CURRENT DEGREE	CURRENT DEGREE[2]	The value of field CURRENT DEGREE on installation panel DSNTIP8	Yes
CURRENT EXPLAIN MODE	Inherited from the invoking application	NO	Yes
CURRENT GET_ACCEL_ARCHIVE	Inherited from the invoking application[6]; otherwise, the subsystem parameter value will be used	The GETACCELARCHIVE bind option value if specified for the user-defined function or stored procedure package; otherwise, the subsystem parameter value will be used	Yes
CURRENT LOCALE LC_CTYPE	Inherited from the invoking application	The value of field CURRENT LC_CTYPE on installation panel DSNTIPF	Yes
CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	Inherited from the invoking application	System default value	Yes

Table 48. Characteristics of special registers in a user-defined function or a stored procedure (continued)			
Special register	**Initial value when INHERIT SPECIAL REGISTERS option is specified**	**Initial value when DEFAULT SPECIAL REGISTERS option is specified**	**Routine can use SET statement to modify?**
CURRENT MEMBER	New value for each SET *host-variable*=CURRENT MEMBER statement	New value for each SET *host-variable*=CURRENT MEMBER statement	Not applicable[5]
CURRENT OPTIMIZATION HINT	The value of bind option OPTHINT for the user-defined function or stored procedure package or inherited from the invoking application[6]	The value of bind option OPTHINT for the user-defined function or stored procedure package	Yes
CURRENT PACKAGE PATH	An empty string if the routine was defined with a COLLID value; otherwise, inherited from the invoking application[4]	An empty string, regardless of whether a COLLID value was specified for the routine[4]	Yes
CURRENT PACKAGESET	Inherited from the invoking application[3]	Inherited from the invoking application[3]	Yes
CURRENT PATH	The value of bind option PATH for the user-defined function or stored procedure package or inherited from the invoking application[6]	The value of bind option PATH for the user-defined function or stored procedure package	Yes
CURRENT PRECISION	Inherited from the invoking application	The value of field DECIMAL ARITHMETIC on installation panel DSNTIP4	Yes
CURRENT QUERY ACCELERATION	Inherited from the invoking application[6]; otherwise, the subsystem parameter value will be used	The QUERYACCELERATION bind option value if specified for the user-defined function or stored procedure package; otherwise, the subsystem parameter value will be used	Yes
CURRENT REFRESH AGE	Inherited from the invoking application	System default value	Yes
CURRENT ROUTINE VERSION	Inherited from the invoking application	The empty string	Yes
CURRENT RULES	Inherited from the invoking application	The value of bind option SQLRULES for the plan that invokes a user-defined function or stored procedure	Yes
CURRENT SCHEMA	Inherited from the invoking application	The value of CURRENT SCHEMA when the routine is entered	Yes
CURRENT SERVER	Inherited from the invoking application	Inherited from the invoking application	Yes

Table 48. Characteristics of special registers in a user-defined function or a stored procedure (continued)

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?
CURRENT SQLID	The primary authorization ID of the application process or inherited from the invoking application[7]	The primary authorization ID of the application process	Yes[8]
CURRENT TEMPORAL BUSINESS_TIME	Inherited from the invoking application	NULL	Yes
CURRENT TEMPORAL SYSTEM_TIME	Inherited from the invoking application	NULL	Yes
CURRENT TIME	New value for each SQL statement in the user-defined function or stored procedure package[1]	New value for each SQL statement in the user-defined function or stored procedure package[1]	Not applicable[5]
CURRENT TIMESTAMP	New value for each SQL statement in the user-defined function or stored procedure package[1]	New value for each SQL statement in the user-defined function or stored procedure package[1]	Not applicable[5]
CURRENT TIMESTAMP WITH TIME ZONE	New value for each SQL statement in the user-defined function or stored procedure package[1]	New value for each SQL statement in the user-defined function or stored procedure package[1]	Not applicable[5]
CURRENT TIME ZONE	Inherited from the invoking application	Inherited from the invoking application	Not applicable[5]
ENCRYPTION PASSWORD	Inherited from the invoking application	Inherited from the invoking application	Yes
SESSION TIME ZONE	Inherited from the invoking application	The value of CURRENT TIME ZONE when the routine is entered	Yes
SESSION_USER or USER	Primary authorization ID of the application process	Primary authorization ID of the application process	Not applicable[5]

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?

Table 48. Characteristics of special registers in a user-defined function or a stored procedure (continued)

Notes:

1. If the user-defined function or stored procedure is invoked within the scope of a trigger, Db2 uses the timestamp for the triggering SQL statement as the timestamp for all SQL statements in the package.
2. Db2 allows parallelism at only one level of a nested SQL statement. If you set the value of the CURRENT DEGREE special register to ANY, and parallelism is disabled, Db2 ignores the CURRENT DEGREE value.
3. If the routine definition includes a specification for COLLID, Db2 sets CURRENT PACKAGESET to the value of COLLID. If both CURRENT PACKAGE PATH and COLLID are specified, the CURRENT PACKAGE PATH value takes precedence and COLLID is ignored.
4. If the function definition includes a specification for PACKAGE PATH, Db2 sets CURRENT PACKAGE PATH to the value of PACKAGE PATH.
5. Not applicable because no SET statement exists for the special register.
6. If a program within the scope of the invoking program issues a SET statement for the special register before the user-defined function or stored procedure is invoked, the special register inherits the value from the SET statement. Otherwise, the special register contains the value that is set by the bind option for the user-defined function or stored procedure package.
7. If a program within the scope of the invoking program issues a SET CURRENT SQLID statement before the user-defined function or stored procedure is invoked, the special register inherits the value from the SET statement. Otherwise, CURRENT SQLID contains the authorization ID of the application process.
8. If the user-defined function or stored procedure package uses a value other than RUN for the DYNAMICRULES bind option, the SET CURRENT SQLID statement can be executed. However, it does not affect the authorization ID that is used for the dynamic SQL statements in the package. The DYNAMICRULES value determines the authorization ID that is used for dynamic SQL statements.

Related concepts

Dynamic rules options for dynamic SQL statements
The DYNAMICRULES bind option and the runtime environment determine the rules for the dynamic SQL attributes.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)
Special registers (Db2 SQL)

Accessing transition tables in a user-defined function or stored procedure

If you want to refer to the entire set of rows that a triggering SQL statement modifies, rather than to individual rows, use a transition table. You can reference a transition table in user-defined functions and procedures that are invoked from a trigger.

About this task

This topic describes how to access transition variables in a user-defined function, but the same techniques apply to a stored procedure.

To access transition tables in a user-defined function, use table locators, which are pointers to the transition tables. You declare table locators as input parameters in the CREATE FUNCTION statement using the TABLE LIKE *table-name* AS LOCATOR clause.

Procedure

To access transition tables in a user-defined function or stored procedure:

1. Declare input parameters to receive table locators. You must define each parameter that receives a table locator as an unsigned 4-byte integer.
2. Declare table locators. You can declare table locators in assembler, C, C++, COBOL, PL/I, and in an SQL procedure compound statement.
3. Declare a cursor to access the rows in each transition table.
4. Assign the input parameter values to the table locators.
5. Access rows from the transition tables using the cursors that are declared for the transition tables.

Results

The following examples show how a user-defined function that is written in C, C++, COBOL, or PL/I accesses a transition table for a trigger. The transition table, NEWEMP, contains modified rows of the employee sample table. The trigger is defined like this:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES (CHECKEMP(TABLE NEWEMPS));
  END;
```

The user-defined function definition looks like this:

```
CREATE FUNCTION CHECKEMP(TABLE LIKE EMP AS LOCATOR)
  RETURNS INTEGER
  EXTERNAL NAME 'CHECKEMP'
  PARAMETER STYLE SQL
  LANGUAGE language;
```

Assembler: The following example shows how an assembler program accesses rows of transition table NEWEMPS.

```
CHECKEMP CSECT
         SAVE  (14,12)             ANY SAVE SEQUENCE
         LR    R12,R15             CODE ADDRESSABILITY
         USING CHECKEMP,R12        TELL THE ASSEMBLER
         LR    R7,R1               SAVE THE PARM POINTER
         USING PARMAREA,R7         SET ADDRESSABILITY FOR PARMS
         USING SQLDSECT,R8         ESTABLISH ADDRESSIBILITY TO SQLDSECT
         L     R6,PROGSIZE         GET SPACE FOR USER PROGRAM
         GETMAIN R,LV=(6)          GET STORAGE FOR PROGRAM VARIABLES
         LR    R10,R1              POINT TO THE ACQUIRED STORAGE
         LR    R2,R10              POINT TO THE FIELD
         LR    R3,R6               GET ITS LENGTH
         SR    R4,R4               CLEAR THE INPUT ADDRESS
         SR    R5,R5               CLEAR THE INPUT LENGTH
         MVCL  R2,R4               CLEAR OUT THE FIELD
         ST    R13,FOUR(R10)       CHAIN THE SAVEAREA PTRS
         ST    R10,EIGHT(R13)      CHAIN SAVEAREA FORWARD
         LR    R13,R10             POINT TO THE SAVEAREA
         USING PROGAREA,R13        SET ADDRESSABILITY
         ST    R6,GETLENTH         SAVE THE LENGTH OF THE GETMAIN
         ⋮
************************************************************
* Declare table locator host variable TRIGTBL              *
************************************************************
TRIGTBL SQL TYPE IS TABLE LIKE EMP AS LOCATOR
************************************************************
* Declare a cursor to retrieve rows from the transition    *
* table                                                    *
************************************************************
         EXEC SQL DECLARE C1 CURSOR FOR                            X
              SELECT LASTNAME FROM TABLE(:TRIGTBL LIKE EMP)        X
              WHERE SALARY > 100000
************************************************************
* Copy table locator for trigger transition table          *
************************************************************
```

```
         L     R2,TABLOC           GET ADDRESS OF LOCATOR
         L     R2,0(0,R2)          GET LOCATOR VALUE
         ST    R2,TRIGTBL
         EXEC SQL OPEN C1
         EXEC SQL FETCH C1 INTO :NAME
         ⋮
         EXEC SQL CLOSE C1
⋮PROGAREA DSECT                     WORKING STORAGE FOR THE PROGRAM
SAVEAREA DS    18F                 THIS ROUTINE'S SAVE AREA
GETLENTH DS    A                   GETMAIN LENGTH FOR THIS AREA
⋮
NAME     DS    CL24
⋮
         DS    0D
PROGSIZE EQU   *-PROGAREA          DYNAMIC WORKAREA SIZE
PARMAREA DSECT
TABLOC   DS    A                   INPUT PARAMETER FOR TABLE LOCATOR
⋮
         END   CHECKEMP
```

C or C++: The following example shows how a C or C++ program accesses rows of transition table NEWEMPS.

```
int CHECK_EMP(int trig_tbl_id)
{
  ⋮
  /*********************************************************/
  /* Declare table locator host variable trig_tbl_id  */
  /*********************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS TABLE LIKE EMP AS LOCATOR trig_tbl_id;
    char name[25];
  EXEC SQL END DECLARE SECTION;
  ⋮
  /*********************************************************/
  /* Declare a cursor to retrieve rows from the transition  */
  /* table                                                  */
  /*********************************************************/
  EXEC SQL DECLARE C1 CURSOR FOR
    SELECT NAME FROM TABLE(:trig_tbl_id LIKE EMPLOYEE)
    WHERE SALARY > 100000;
  /*********************************************************/
  /* Fetch a row from transition table                      */
  /*********************************************************/
  EXEC SQL OPEN C1;
  EXEC SQL FETCH C1 INTO :name;
  ⋮
  EXEC SQL CLOSE C1;
  ⋮
}
```

COBOL: The following example shows how a COBOL program accesses rows of transition table NEWEMPS.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CHECKEMP.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  NAME   PIC X(24).
       ⋮
       LINKAGE SECTION.
      *********************************************************
      * Declare table locator host variable TRIG-TBL-ID       *
      *********************************************************
       01  TRIG-TBL-ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR.
       ⋮
       PROCEDURE DIVISION USING TRIG-TBL-ID.
       ⋮
      *********************************************************
      * Declare cursor to retrieve rows from transition table *
      *********************************************************
       EXEC SQL DECLARE C1 CURSOR FOR
         SELECT NAME FROM TABLE(:TRIG-TBL-ID LIKE EMP)
         WHERE SALARY > 100000 END-EXEC.
      *********************************************************
      * Fetch a row from transition table                     *
      *********************************************************
```

```
       EXEC SQL OPEN C1 END-EXEC.
       EXEC SQL FETCH C1 INTO :NAME END-EXEC.
       ⋮
       EXEC SQL CLOSE C1 END-EXEC.
       ⋮
       PROG-END.
       GOBACK.
```

PL/I: The following example shows how a PL/I program accesses rows of transition table NEWEMPS.

```
CHECK_EMP: PROC(TRIG_TBL_ID) RETURNS(BIN FIXED(31))
           OPTIONS(MAIN NOEXECOPS REENTRANT);
/****************************************************/
/* Declare table locator host variable TRIG_TBL_ID  */
/****************************************************/
DECLARE TRIG_TBL_ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR;
DECLARE NAME CHAR(24);
⋮
/****************************************************/
/* Declare a cursor to retrieve rows from the       */
/* transition table                                 */
/****************************************************/
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:TRIG_TBL_ID LIKE EMP)
  WHERE SALARY > 100000;
/****************************************************/
/* Retrieve rows from the transition table          */
/****************************************************/
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :NAME;
⋮
EXEC SQL CLOSE C1;
⋮
END CHECK_EMP;
```

Preparing an external user-defined function for execution

Because an external user-defined function is written in a programming language, preparing it is similar to the way that you prepare any other application program.

Procedure

To prepare an external user-defined function for execution:

1. Precompile the user-defined function program and bind the DBRM into a package. You need to do this only if your user-defined function contains SQL statements. You do not need to bind a plan for the user-defined function.
2. Compile the user-defined function program and link-edit it with Language Environment and RRSAF.

   You must compile the program with a compiler that supports Language Environment and link-edit the appropriate Language Environment components with the user-defined function. You must also link-edit the user-defined function with RRSAF.

   The program preparation JCL samples DSNHASM, DSNHC, DSNHCPP, DSNHICOB, and DSNHPLI show you how to precompile, compile, and link-edit assembler, C, C++, COBOL, and PL/I Db2 programs. For object-oriented programs in C++, see JCL sample DSNHCPP2 for program preparation hints.
3. For a user-defined function that contains SQL statements, grant EXECUTE authority on the user-defined function package to the function definer.

Abnormal termination of an external user-defined function

If an external user-defined function abnormally terminates, your program receives SQLCODE -430 for invoking the statement.

Db2 also performs the following actions:

- Places the unit of work that contains the invoking statement in a must-rollback state.
- Stops the user-defined function, and subsequent calls fail, in either of the following situations:

- The number of abnormal terminations equals the STOP AFTER *n* FAILURES value for the user-defined function.
- If the STOP AFTER *n* FAILURES option is not specified, the number of abnormal terminations equals the default MAX ABEND COUNT value for the subsystem.

You should include code in your program to check for a user-defined function abend and to roll back the unit of work that contains the user-defined function invocation.

Saving information between invocations of a user-defined function by using a scratchpad

If you create a scratchpad for a reentrant user-defined function, Db2 can use it to preserve information between invocations of the function.

About this task

You can use a scratchpad to save information between invocations of a user-defined function. To indicate that a scratchpad should be allocated when the user-defined function executes, the function definer specifies the SCRATCHPAD parameter in the CREATE FUNCTION statement.

The scratchpad consists of a 4-byte length field, followed by the scratchpad area. The definer can specify the length of the scratchpad area in the CREATE FUNCTION statement. The specified length does not include the length field. The default size is 100 bytes. Db2 initializes the scratchpad for each function to binary zeros at the beginning of execution for each subquery of an SQL statement and does not examine or change the content thereafter. On each invocation of the user-defined function, Db2 passes the scratchpad to the user-defined function. You can therefore use the scratchpad to preserve information between invocations of a reentrant user-defined function.

The following example demonstrates how to enter information in a scratchpad for a user-defined function defined like this:

```
CREATE FUNCTION COUNTER()
  RETURNS INT
  SCRATCHPAD
  FENCED
  NOT DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE SQL
  EXTERNAL NAME 'UDFCTR';
```

The scratchpad length is not specified, so the scratchpad has the default length of 100 bytes, plus 4 bytes for the length field. The user-defined function increments an integer value and stores it in the scratchpad on each execution.

```
#pragma linkage(ctr,fetchable)
#include <stdlib.h>
#include <stdio.h>
/* Structure scr defines the passed scratchpad for function ctr */
  struct scr {
    long len;
    long countr;
    char not_used[96];
  };
/***************************************************************/
/* Function ctr:  Increments a counter and reports the value   */
/*                from the scratchpad.                         */
/*                                                             */
/*     Input:  None                                            */
/*     Output: INTEGER out      the value from the scratchpad  */
/***************************************************************/
void ctr(
  long *out,                        /* Output answer (counter)  */
  short *outnull,                   /* Output null indicator    */
  char *sqlstate,                   /* SQLSTATE                 */
  char *funcname,                   /* Function name            */
  char *specname,                   /* Specific function name   */
```

```
  char *mesgtext,                 /* Message text insert      */
  struct scr *scratchptr)         /* Scratchpad               */
{
  *out = ++scratchptr->countr;    /* Increment counter and    */
                                  /* copy to output variable  */
  *outnull = 0;                   /* Set output null indicator*/
   return;
}
/* end of user-defined function ctr */
```

Example of creating and using a user-defined scalar function

You can create a user-defined scalar function that gets input from a table and puts the output in a table.

Suppose that your organization needs a user-defined scalar function that calculates the bonus that each employee receives. All employee data, including salaries, commissions, and bonuses, is kept in the employee table, EMP. The input fields for the bonus calculation function are the values of the SALARY and COMM columns. The output from the function goes into the BONUS column. Because this function gets its input from a Db2 table and puts the output in a Db2 table, a convenient way to manipulate the data is through a user-defined function.

The user-defined function's definer and invoker determine that this new user-defined function should have these characteristics:

- The user-defined function name is CALC_BONUS.
- The two input fields are of type DECIMAL(9,2).
- The output field is of type DECIMAL(9,2).
- The program for the user-defined function is written in COBOL and has a load module name of CBONUS.

Because no built-in function or user-defined function exists on which to build a sourced user-defined function, the function implementer must code an external user-defined function. The implementer performs the following steps:

- Writes the user-defined function, which is a COBOL program
- Precompiles, compiles, and links the program
- Binds a package if the user-defined function contains SQL statements
- Tests the program thoroughly
- Grants execute authority on the user-defined function package to the definer

The user-defined function definer executes this CREATE FUNCTION statement to register CALC_BONUS to Db2:

```
CREATE FUNCTION CALC_BONUS(DECIMAL(9,2),DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  EXTERNAL NAME 'CBONUS'
  PARAMETER STYLE SQL
  LANGUAGE COBOL;
```

The definer then grants execute authority on CALC_BONUS to all invokers.

User-defined function invokers write and prepare application programs that invoke CALC_BONUS. An invoker might write a statement like this, which uses the user-defined function to update the BONUS field in the employee table:

```
UPDATE EMP
  SET BONUS = CALC_BONUS(SALARY,COMM);
```

An invoker can execute this statement either statically or dynamically.

User-defined function samples that ship with Db2

To assist you in defining, implementing, and invoking your user-defined functions, Db2 provides a number of sample user-defined functions. All sample user-defined function code is in data set DSN1110.SDSNSAMP.

The following table summarizes the characteristics of the sample user-defined functions.

Table 49. User-defined function samples shipped with Db2

User-defined function name	Language	Member that contains source code	Purpose
ALTDATE[1]	C	DSN8DUAD	Converts the current date to a user-specified format
ALTDATE[2]	C	DSN8DUCD	Converts a date from one format to another
ALTTIME[3]	C	DSN8DUAT	Converts the current time to a user-specified format
ALTTIME[4]	C	DSN8DUCT	Converts a time from one format to another
DAYNAME	C++	DSN8EUDN	Returns the day of the week for a user-specified date
HDFS_READ	C++	DSN8HDFS	Reads data from a delimiter-separated file in the Hadoop Distributed File System (HDFS)
JAQL_SUBMIT	C++	DSN8JAQL	Invokes an IBM InfoSphere® BigInsights® Jaql query
MONTHNAME	C++	DSN8EUMN	Returns the month for a user-specified date
CURRENCY	C	DSN8DUCY	Formats a floating-point number as a currency value
TABLE_NAME	C	DSN8DUTI	Returns the unqualified table name for a table, view, or alias
TABLE_QUALIF	C	DSN8DUTI	Returns the qualifier for a table, view, or alias
TABLE_LOCATION	C	DSN8DUTI	Returns the location for a table, view, or alias
WEATHER	C	DSN8DUWF	Returns a table of weather information from a EBCDIC data set

Notes:

1. This version of ALTDATE has one input parameter, of type VARCHAR(13).
2. This version of ALTDATE has three input parameters, of type VARCHAR(17), VARCHAR(13), and VARCHAR(13).
3. This version of ALTTIME has one input parameter, of type VARCHAR(14).
4. This version of ALTTIME has three input parameters, of type VARCHAR(11), VARCHAR(14), and VARCHAR(14).

Member DSN8DUWC contains a client program that shows you how to invoke the WEATHER user-defined table function.

Member DSNTEJBI shows you how to define and prepare the IBM InfoSphere BigInsights sample user-defined functions.

Member DSNTEJ2U shows you how to define and prepare the other sample user-defined functions and the client program.

Related concepts
Job DSNTEJBI (Db2 Installation and Migration)
Job DSNTEJ2U (Db2 Installation and Migration)
Sample user-defined functions (Db2 SQL)

Creating stored procedures

A *stored procedure* is executable code that can be called by other programs. The process for creating one depends on the type of procedure.

Before you begin

You must complete some configuration tasks for the Db2 environment before you can use any of the following types of procedures:

- External stored procedures
- Native SQL procedures that satisfy any of the following conditions:
  - Calls at least one external stored procedure, external SQL procedure, or user-defined function.
  - Defined with ALLOW DEBUG MODE or DISALLOW DEBUG MODE.
- External SQL procedures (deprecated)
- Db2-supplied stored procedures

For instructions, see Installation step 19: Configure Db2 for running stored procedures and user-defined functions (Db2 Installation and Migration) or Migration step 22: Configure Db2 for running stored procedures and user-defined functions (optional) (Db2 Installation and Migration).

Procedure

Follow the process for the type of stored procedure that you want to create, and issue a CREATE PROCEDURE statement to register the stored procedure with a database server.

You can create the following types of stored procedures:

Native SQL procedures

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is contained and specified in the procedure definition along with various attributes of the procedure. A package is generated for a native SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2. Each time that the procedure is invoked, the package executes one or more times.

All SQL procedures that are created with a CREATE PROCEDURE statement that does not specify the FENCED or EXTERNAL options are native SQL procedures. More capabilities are supported for native SQL procedures, they usually perform better than external SQL procedures, and no associated C program is generated for them.

For more information, see "Creating native SQL procedures" on page 231.

External stored procedures

The procedure body is an external program that is written in a programming language such as C, C++, COBOL, or Java and it can contain SQL statements. The source code for an external stored procedure is separate from the procedure definition and is bound into a package. The name of the external executable is specified as part of the procedure definition along with various attributes of the procedure. All programs must be designed to run using Language Environment. Your COBOL and C++ stored procedures can contain object-oriented extensions. Each time that the stored procedure is invoked, the logic in the procedure controls whether the package executes and how many times.

For more information, see "Creating external stored procedures" on page 256.

External SQL procedures (deprecated)

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is specified in the procedure definition along with various attributes of the procedure. A C program and an associated package are generated for an external SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2.Each time that the procedure is invoked, the package executes one or more times.

Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

For more information, see "Creating external SQL procedures (deprecated)" on page 291.

GUPI

Related concepts

Stored procedures

A *stored procedure* is a compiled program that can execute SQL statements and is stored at a local or remote Db2 server. You can invoke a stored procedure from an application program or from the command line processor. A single call to a stored procedure from a client application can access the database at the server several times.

External stored procedures

An *external stored procedure* is a procedure that is written in a host language and can contain SQL statements. The source code for external procedures is separate from the definition.

SQL procedures

An SQL procedure is a stored procedure that contains only SQL statements.

Related reference

CREATE PROCEDURE (Db2 SQL)

Db2 for z/OS Exchange

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Stored procedures

A *stored procedure* is a compiled program that can execute SQL statements and is stored at a local or remote Db2 server. You can invoke a stored procedure from an application program or from the command line processor. A single call to a stored procedure from a client application can access the database at the server several times.

A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a host language or SQL procedure statements. You can call stored procedures from other applications or from the command line. Db2 provides some stored procedures, but you can also create your own.

A stored procedure provides a common piece of code that is written only once and is maintained in a single instance that can be called from several different applications. Host languages can easily call procedures that exist on a local system, and SQL can call stored procedures that exist on remote systems. In fact, a major benefit of procedures in SQL is that they can be used to enhance the performance characteristics of distributed applications. With stored procedures, you can avoid network transfer of large amounts of data obtained as part of intermediate results in a long sequence of queries.

The following diagram illustrates the processing for an application that does not use stored procedures. The client application embeds SQL statements and communicates with the server separately for each statement. This application design results in increased network traffic and processor costs.

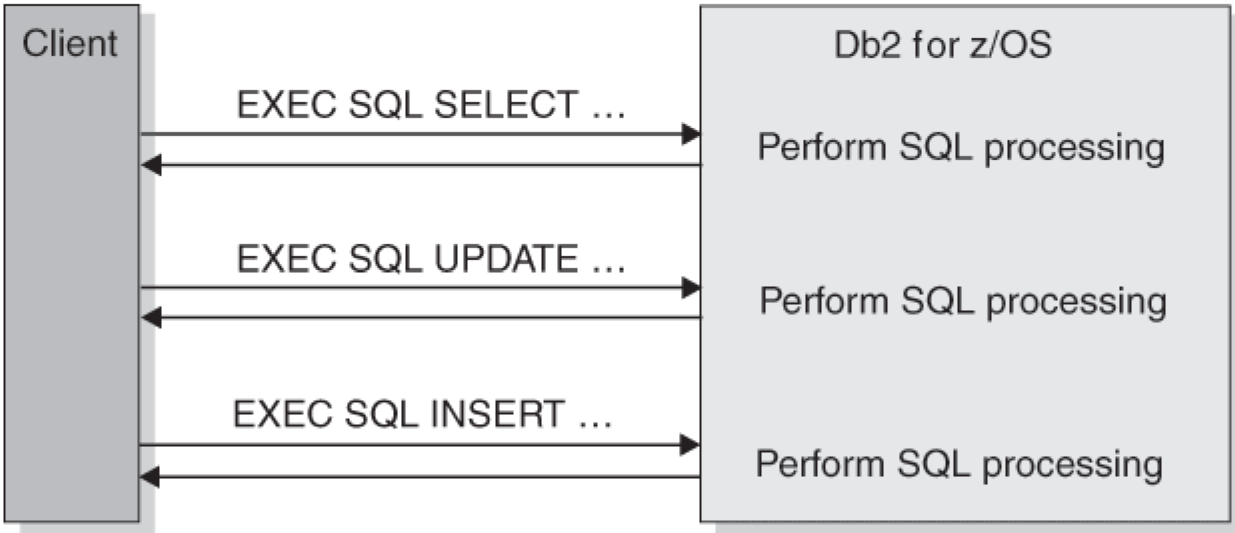


Figure 9. Processing without stored procedures

The following diagram illustrates the processing for an application that uses stored procedures. Because a stored procedure is used on the server, a series of SQL statements can be executed with a single send and receive operation, reducing network traffic and the cost of processing these statements.

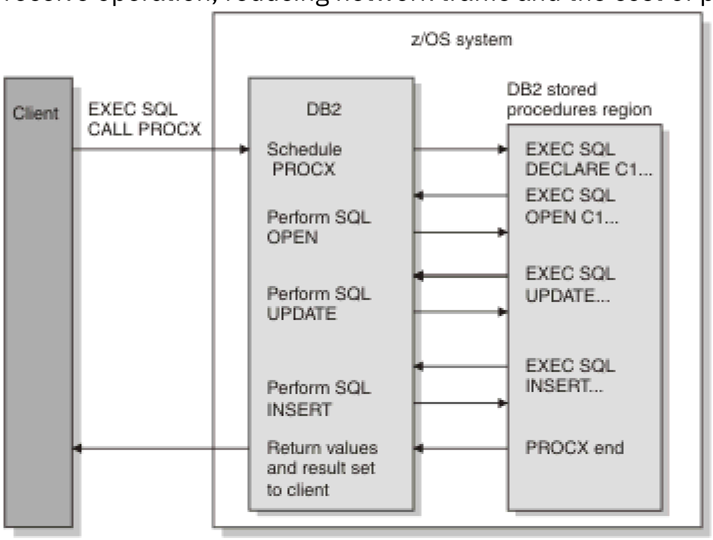


Figure 10. Processing with stored procedures

Stored procedures are useful for client/server applications that do at least one of the following things:

- Execute multiple remote SQL statements. Remote SQL statements can create many network send and receive operations, which results in increased processor costs. Stored procedures can encapsulate many of your application's SQL statements into a single message to the Db2 server, reducing network traffic to a single send and receive operation for a series of SQL statements. Locks on Db2 tables are not held across network transmissions, which reduces contention for resources at the server.
- Access tables from a dynamic SQL environment where table privileges for the application that is running are undesirable. Stored procedures allow static SQL authorization from a dynamic environment.
- Access host variables for which you want to guarantee security and integrity. Stored procedures remove SQL applications from the workstation, which prevents workstation users from manipulating the contents of sensitive SQL statements and host variables.
- Create a result set of rows to return to the client application.

Stored procedures that are written in embedded static SQL provide the following additional advantages:

- Better performance because static SQL is prepared at precompile time and has no run time overhead for access plan (package) generation.
- Encapsulation enables programmers to write applications that access data without knowing the details of database objects.
- Improved security because access privileges are encapsulated within the packages that are associated with the stored procedures. You can grant access to run a stored procedure that selects data from tables, without granting SELECT privilege to the user.

You can create the following types of stored procedures:

Native SQL procedures

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is contained and specified in the procedure definition along with various attributes of the procedure. A package is generated for a native SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2. Each time that the procedure is invoked, the package executes one or more times.

All SQL procedures that are created with a CREATE PROCEDURE statement that does not specify the FENCED or EXTERNAL options are native SQL procedures. More capabilities are supported for native SQL procedures, they usually perform better than external SQL procedures, and no associated C program is generated for them.

For more information, see "Creating native SQL procedures" on page 231.

External stored procedures

The procedure body is an external program that is written in a programming language such as C, C++, COBOL, or Java and it can contain SQL statements. The source code for an external stored procedure is separate from the procedure definition and is bound into a package. The name of the external executable is specified as part of the procedure definition along with various attributes of the procedure. All programs must be designed to run using Language Environment. Your COBOL and C++ stored procedures can contain object-oriented extensions. Each time that the stored procedure is invoked, the logic in the procedure controls whether the package executes and how many times.

For more information, see "Creating external stored procedures" on page 256.

External SQL procedures (deprecated)

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is specified in the procedure definition along with various attributes of the procedure. A C program and an associated package are generated for an external SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2.Each time that the procedure is invoked, the package executes one or more times.

Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

For more information, see "Creating external SQL procedures (deprecated)" on page 291.

Db2 also provides a set of stored procedures that you can call in your application programs to perform a number of utility, application programming, and performance management functions. These procedures are called *supplied stored procedures*. Typically, you create these procedures during installation or migration.

Difference between native SQL and external procedures

Related concepts

Common SQL API stored procedures (Db2 Administration Guide)

Related tasks

Implementing Db2 stored procedures (Stored procedures provided by Db2)

Related reference

Procedures that are supplied with Db2 (Db2 SQL)

Stored procedure parameters

You can pass information between a stored procedure and the calling application program by using parameters. Applications pass the required parameters in the SQL CALL statement. Optionally, the application can also include an indicator variable with each parameter to allow for null values or to pass large output parameter values.

You define the stored procedure parameters as part of the stored procedure definition in the CREATE PROCEDURE statement. The stored procedure parameters can be one of the following types:

IN
: Input-only parameters, which provide values to the stored procedure.

OUT
: Output-only parameters, which return values from the stored procedure to the calling program.

INOUT
: Input and output parameters, which provide values to and return values from the stored procedure.

If a stored procedure fails to set one or more of the OUT or INOUT parameters, Db2 does not return an error. Instead, Db2 returns the output parameters to the calling program, with the values that were established on entry to the stored procedure.

Within a procedure body, the following rules apply to IN, OUT, and INOUT parameters:

- You can use a parameter that you define as IN on the left side or right side of an assignment statement. However, if you assign a value to an IN parameter, you cannot pass the new value back to the caller. The IN parameter has the same value before and after the SQL procedure is called.
- You can use a parameter that you define as OUT on the left side or right side of an assignment statement. The last value that you assign to the parameter is the value that is returned to the caller. The starting value of an OUT parameter is NULL.
- You can use a parameter that you define as INOUT on the left side or right side of an assignment statement. The caller determines the first value of the INOUT parameter, and the last value that you assign to the parameter is the value that is returned to the caller.

Restrictions:

- You cannot pass file reference variables as stored procedure parameters.
- You cannot pass parameters with the type XML to stored procedures. You can specify tables or views that contain XML columns as table locator parameters. However, you cannot reference the XML columns in the body of the stored procedure.

Related tasks

Calling a stored procedure from your application
To run a stored procedure, you can either call it from a client program or invoke it from the command line processor.

Passing large output parameters to stored procedures by using indicator variables
If any output parameters occupy a large amount of storage, passing the entire storage area to a stored procedure can degrade performance.

Related reference

CALL (Db2 SQL)
CREATE PROCEDURE (Db2 SQL)

Example of a simple stored procedure

When an application that runs on a workstation calls a stored procedure on a Db2 server, the stored procedure updates a table based on the information that it receives from the application.

Suppose that an application runs on a workstation client and calls a stored procedure A on the Db2 server at location LOCA. Stored procedure A performs the following operations:

1. Receives a set of parameters containing the data for one row of the employee to project activity table (DSN8B10.EMPPROJACT). These parameters are input parameters in the SQL statement CALL:

- EMP: employee number
- PRJ: project number
- ACT: activity ID
- EMT: percent of employee's time required
- EMS: date the activity starts
- EME: date the activity is due to end

2. Declares a cursor, C1, with the option WITH RETURN, that is used to return a result set containing all rows in EMPPROJACT to the workstation application that called the stored procedure.
3. Queries table EMPPROJACT to determine whether a row exists where columns PROJNO, ACTNO, EMSTDATE, and EMPNO match the values of parameters PRJ, ACT, EMS, and EMP. (The table has a unique index on those columns. There is at most one row with those values.)
4. If the row exists, executes an SQL statement UPDATE to assign the values of parameters EMT and EME to columns EMPTIME and EMENDATE.[1]
5. If the row does not exist (SQLCODE +100), executes an SQL statement INSERT to insert a new row with all the values in the parameter list.[1]
6. Opens cursor C1. This causes the result set to be returned to the caller when the stored procedure ends.
7. Returns two parameters, containing these values:
   - A code to identify the type of SQL statement last executed: UPDATE or INSERT.
   - The SQLCODE from that statement.

Note:

1. Alternatively, steps 4 and 5 can be accomplished with a single MERGE statement.

The following figure illustrates the steps that are involved in executing this stored procedure.

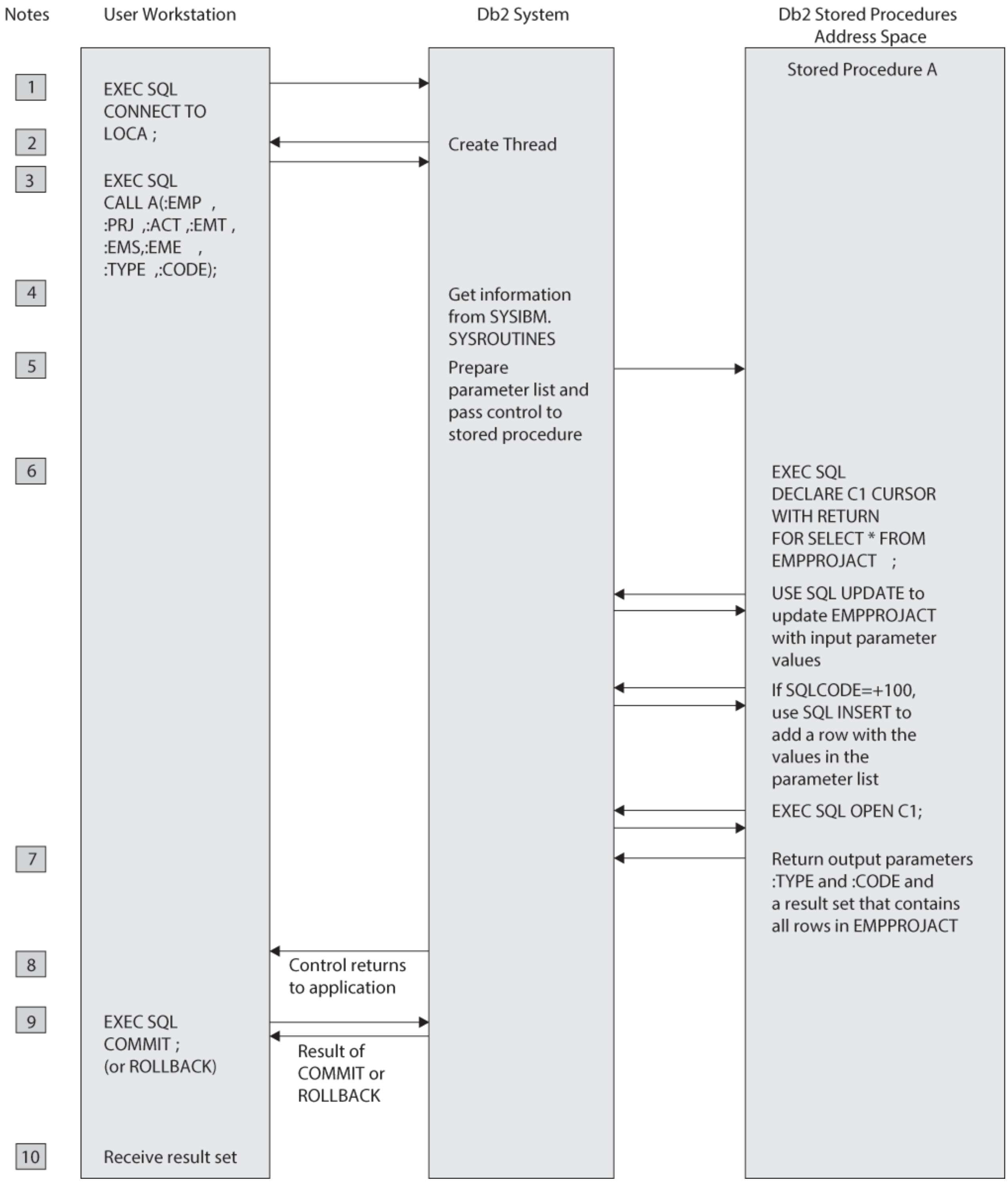


Figure 11. Stored procedure overview

Notes:

1. The workstation application uses the SQL CONNECT statement to create a conversation with Db2.
2. Db2 creates a Db2 thread to process SQL requests.
3. The SQL statement CALL tells the Db2 server that the application is going to run a stored procedure. The calling application provides the necessary parameters.
4. The plan for the client application contains information from catalog table SYSIBM.SYSROUTINES about stored procedure A.

5. Db2 passes information about the request to the stored procedures address space, and the stored procedure begins execution.
6. The stored procedure executes SQL statements.

   Db2 verifies that the owner of the package or plan containing the SQL statement CALL has EXECUTE authority for the package associated with the Db2 stored procedure.

   One of the SQL statements opens a cursor that has been declared WITH RETURN. This causes a result set to be returned to the workstation application when the procedure ends.

   Any SQLCODE that is issued within an **external** stored procedure is **not** returned to the workstation application in the SQLCA (as the result of the CALL statement).
7. If an error is not encountered, the stored procedure assigns values to the output parameters and exits.

   Control returns to the Db2 stored procedures address space, and from there to the Db2 system. If the stored procedure definition contains COMMIT ON RETURN NO, Db2 does not commit or roll back any changes from the SQL in the stored procedure until the calling program executes an explicit COMMIT or ROLLBACK statement. If the stored procedure definition contains COMMIT ON RETURN YES, and the stored procedure executed successfully, Db2 commits all changes. The COMMIT statement closes the cursor unless it is declared with the WITH HOLD option.
8. Control returns to the calling application, which receives the output parameters and the result set. Db2 then:

   - Closes all cursors that the stored procedure opened, except those that the stored procedure opened to return result sets.
   - Discards all SQL statements that the stored procedure prepared.
   - Reclaims the working storage that the stored procedure used.

   The application can call more stored procedures, or it can execute more SQL statements. Db2 receives and processes the COMMIT or ROLLBACK request. The COMMIT or ROLLBACK operation covers all SQL operations, whether executed by the application or by stored procedures, for that unit of work.

   If the application involves IMS or CICS, similar processing occurs based on the IMS or CICS sync point rather than on an SQL COMMIT or ROLLBACK statement.
9. Db2 returns a reply message to the application describing the outcome of the COMMIT or ROLLBACK operation.
10. The workstation application executes the following steps to retrieve the contents of table EMPPROJACT, which the stored procedure has returned in a result set:

    a. Declares a result set locator for the result set being returned.
    b. Executes the ASSOCIATE LOCATORS statement to associate the result set locator with the result set.
    c. Executes the ALLOCATE CURSOR statement to associate a cursor with the result set.
    d. Executes the FETCH statement with the allocated cursor multiple times to retrieve the rows in the result set.
    e. Executes the CLOSE statement to close the cursor.

SQL procedures

An SQL procedure is a stored procedure that contains only SQL statements.

The source code for these procedures (the SQL statements) is specified in CREATE PROCEDURE statement. The part of the CREATE PROCEDURE statement that contains SQL statements is called the *procedure body*.

Types of SQL procedures

Db2 for z/OS supports the following types of SQL procedures:

Native SQL procedures

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is contained and specified in the procedure definition along with various attributes of the procedure. A package is generated for a native SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2. Each time that the procedure is invoked, the package executes one or more times.

All SQL procedures that are created with a CREATE PROCEDURE statement that does not specify the FENCED or EXTERNAL options are native SQL procedures. More capabilities are supported for native SQL procedures, they usually perform better than external SQL procedures, and no associated C program is generated for them.

For more information, see "Creating native SQL procedures" on page 231.

External SQL procedures (deprecated)

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is specified in the procedure definition along with various attributes of the procedure. A C program and an associated package are generated for an external SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2.Each time that the procedure is invoked, the package executes one or more times.

Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

For more information, see "Creating external SQL procedures (deprecated)" on page 291.

Native SQL procedures

A *native SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). A native SQL procedure is created by issuing a single SQL statement, CREATE PROCEDURE. Native SQL procedures do not require any other program preparation, such as precompiling, compiling, or link-editing source code. Native SQL procedures are executed as SQL statements that are bound in a Db2 package. Native SQL procedures do not have an associated external application program. Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

Native SQL procedures have the following advantages:

- You can create them in one step.
- They do not run in a WLM environment.
- They might be eligible for zIIP redirect if they are invoked remotely through a DRDA client.
- They usually perform better than external SQL procedures.
- They support more capabilities, such as nested compound statements, than external SQL procedures.
- Db2 can manage multiple versions of these procedures for you.
- You can specify that the SQL procedure commits autonomously, without committing the work of the calling application.

All SQL procedures that are created without the FENCED or EXTERNAL options in the CREATE PROCEDURE statement are native SQL procedures.

External SQL procedures (deprecated)

An *external SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). However, an external SQL procedure is created, implemented, and executed like other external stored procedures.

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

All SQL procedures that were created prior to DB2 9 are external SQL procedures. Starting in Version DB2 9, you can create an external SQL procedure by specifying FENCED or EXTERNAL in the CREATE PROCEDURE statement.

SQL procedure body
The body of an SQL procedure contains one or more SQL statements. In the SQL procedure body, you can also declare and use variables, conditions, return codes, statements, cursors, and handlers.

Statements that you can include in an SQL procedure body

A CREATE PROCEDURE statement for a native SQL procedure contains an *SQL-routine-body*, as defined in CREATE PROCEDURE (SQL - native) (Db2 SQL). The syntax diagram for *SQL-routine-body* defines the procedure body as a single SQL statement. The SQL statement can be one of the SQL statements that are shown in the syntax diagram for *SQL-routine-body*, or an SQL control statement. The syntax diagram for *SQL-control-statement* in SQL procedural language (SQL PL) (Db2 SQL) identifies the control statements that can be specified.

A native SQL procedure can contain multiple SQL statements if the outermost SQL statement is an *SQL-control-statement* that includes other SQL statements. These statements are defined as SQL procedure statements. The syntax diagram in SQL-procedure-statement (Db2 SQL) identifies the SQL statements that can be specified within a control statement. The syntax notes for *SQL-procedure-statement* clarify the SQL statements that are allowed in a native SQL procedure.

Examples

The following examples show how to determine whether an SQL statement is allowed in an SQL procedure.

The syntax diagrams for the control statements indicate where semicolons are needed in an SQL procedure. If the procedure contains a single statement that is not a control statement, such as Example 1, then no semicolons are in the CREATE PROCEDURE statement. If the procedure consists of multiple statements, such as Example 2, use semicolons to separate SQL statements within the SQL procedure. Do not put a semicolon after the outermost control statement.

Example 1

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL
 MODIFIES SQL DATA
 DETERMINISTIC
 COMMIT ON RETURN YES
   UPDATE EMP  A
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPLOYEE_NUMBER
```

The UPDATE statement (A) is an SQL statement that is allowed because it is listed in the syntax diagram for *SQL-routine-body*.

Example 2

```
CREATE PROCEDURE GETWEEKENDS(IN MYDATES DATEARRAY, OUT WEEKENDS DATEARRAY)
 BEGIN  A
  -- ARRAY INDEX VARIABLES
  DECLARE DATEINDEX, WEEKENDINDEX INT DEFAULT 1;  B
  -- VARIABLE TO STORE THE ARRAY LENGTH OF MYDATES,
  -- INITIALIZED USING THE CARDINALITY FUNCTION.
  DECLARE DATESCOUNT INT;  B
```

```
   SET DATESCOUNT = CARDINALITY(MYDATES); C
   -- FOR EACH DATE IN MYDATES, IF THE DATE IS A SUNDAY OR SATURDAY,
   -- ADD IT TO THE OUTPUT ARRAY NAMED "WEEKENDS"
   WHILE DATEINDEX <= DATESCOUNT DO D
    IF DAYOFWEEK(MYDATES[DATEINDEX]) IN (1, 7) THEN E
     SET WEEKENDS[WEEKENDINDEX] = MYDATES[DATEINDEX]; C
     SET WEEKENDINDEX = WEEKENDINDEX + 1; C
    END IF;
    SET DATEINDEX = DATEINDEX + 1; C
   END WHILE;
  END A
```

The SQL procedure has the following keywords and statements:

- The BEGIN and END keywords (A) indicate the beginning and the end of a compound statement.
- The DECLARE statements (B) are components of a compound statement, and define SQL variables within the compound statement.
- The SET assignment statements (C) are SQL control statements that assign values to SQL variables.
- The WHILE statement (D) and the IF statement (E) are SQL control statements.

A compound statement is an SQL control statement. SQL control statements are allowed in the SQL procedure body because *SQL-control-statement* is listed in the syntax diagram for *SQL-routine-body* of a CREATE PROCEDURE (SQL - native) statement.

Related concepts

Nested compound statements in native SQL procedures
Nested compound statements are blocks of SQL statements that are contained by other blocks of SQL statements in native SQL procedures. Use nested compound statements to define condition handlers that execute more than one statement and to define different scopes for variables and condition handlers.

Stored procedure parameters
You can pass information between a stored procedure and the calling application program by using parameters. Applications pass the required parameters in the SQL CALL statement. Optionally, the application can also include an indicator variable with each parameter to allow for null values or to pass large output parameter values.

Promotion of data types (Db2 SQL)

Related reference

compound-statement (Db2 SQL)

Variables in SQL procedures

For data that you use only within an SQL procedure, you can declare *SQL variables* and store the values in the variables. SQL variables are similar to host variables in external stored procedures. SQL variables can be defined with the same data types and lengths as SQL procedure parameters.

An SQL variable declaration has the following form:

```
DECLARE SQL-variable-name data-type;
```

An SQL variable is defined in a compound statement. SQL variables can be referenced anywhere in the compound statement in which they are declared, including any SQL statement that is directly or indirectly nested within that compound statement. For more information, see References to SQL parameters and variables (Db2 SQL).

You can perform any operations on SQL variables that you can perform on host variables in SQL statements.

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Using host variables in SQL statements

Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Related tasks

Controlling the scope of variables in an SQL procedure
Use nested compound statements within an SQL procedure to define the scope of SQL variables. You can reference the variable only within the compound statement in which it was declared and within any nested statements.

Examples of SQL procedures
You can use CASE statements, compound statements, and nested statements within an SQL procedure body.

Example: CASE statement: The following SQL procedure demonstrates how to use a CASE statement. The procedure receives an employee's ID number and rating as input parameters. The CASE statement modifies the employee's salary and bonus, using a different UPDATE statement for each of the possible ratings.

```
CREATE PROCEDURE UPDATESALARY2
 (IN EMPNUMBR CHAR(6),
  IN RATING INT)
 LANGUAGE SQL
 MODIFIES SQL DATA
 CASE RATING
  WHEN 1 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.10, BONUS = 1000
    WHERE EMPNO = EMPNUMBR;
  WHEN 2 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.05, BONUS = 500
    WHERE EMPNO = EMPNUMBR;
  ELSE
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.03, BONUS = 0
    WHERE EMPNO = EMPNUMBR;
 END CASE
```

Example: Compound statement with nested IF and WHILE statements: The following example shows a compound statement that includes an IF statement, a WHILE statement, and assignment statements. The example also shows how to declare SQL variables, cursors, and handlers for classes of error codes.

The procedure receives a department number as an input parameter. A WHILE statement in the procedure body fetches the salary and bonus for each employee in the department, and uses an SQL variable to calculate a running total of employee salaries for the department. An IF statement within the WHILE statement tests for positive bonuses and increments an SQL variable that counts the number of bonuses in the department. When all employee records in the department have been processed, a NOT FOUND condition occurs. A NOT FOUND condition handler makes the search condition for the WHILE statement false, so execution of the WHILE statement ends. Assignment statements then assign the total employee salaries and the number of bonuses for the department to the output parameters for the stored procedure.

If any SQL statement in the compound statement P1 receives an error, the SQLEXCEPTION handler receives control. The handler action sets the output parameter DEPTSALARY to NULL. After the handler action has completed successfully, the original error condition is resolved (SQLSTATE '00000', SQLCODE 0). Because this handler is an EXIT handler, execution passes to the end of the compound statement, and the SQL procedure ends.

```
CREATE PROCEDURE RETURNDEPTSALARY
 (IN DEPTNUMBER CHAR(3),
  OUT DEPTSALARY DECIMAL(15,2),
  OUT DEPTBONUSCNT INT)
 LANGUAGE SQL
 READS SQL DATA
 P1: BEGIN
     DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
     DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
     DECLARE TOTAL_SALARY DECIMAL(15,2) DEFAULT 0;
```

```
     DECLARE BONUS_CNT INT DEFAULT 0;
     DECLARE END_TABLE INT DEFAULT 0;
     DECLARE C1 CURSOR FOR
      SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = DEPTNUMBER;
     DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET END_TABLE = 1;
     DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET DEPTSALARY = NULL;
     OPEN C1;
     FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
     WHILE END_TABLE = 0 DO
      SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
      IF EMPLOYEE_BONUS > 0 THEN
       SET BONUS_CNT = BONUS_CNT + 1;
      END IF;
      FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
     END WHILE;
     CLOSE C1;
     SET DEPTSALARY = TOTAL_SALARY;
     SET DEPTBONUSCNT = BONUS_CNT;
 END P1
```

Example: Compound statement with dynamic SQL statements: The following example shows a compound statement that includes dynamic SQL statements.

The procedure receives a department number (P_DEPT) as an input parameter. In the compound statement, three statement strings are built, prepared, and executed:

- The first statement string executes a DROP statement to ensure that the table to be created does not already exist. This table is named DEPT_*deptno*_T, where *deptno* is the value of input parameter P_DEPT.
- The next statement string executes a CREATE statement to create DEPT_*deptno*_T.
- The third statement string inserts rows for employees in department *deptno* into DEPT_*deptno*_T.

Just as statement strings that are prepared in host language programs cannot contain host variables, statement strings in SQL procedures cannot contain SQL variables or stored procedure parameters. Therefore, the third statement string contains a parameter marker that represents P_DEPT. When the prepared statement is executed, parameter P_DEPT is substituted for the parameter marker.

```
CREATE PROCEDURE CREATEDEPTTABLE (IN P_DEPT CHAR(3))
 LANGUAGE SQL
 BEGIN
  DECLARE STMT CHAR(1000);
  DECLARE MESSAGE CHAR(20);
  DECLARE TABLE_NAME CHAR(30);
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
   SET MESSAGE = 'ok';
  SET TABLE_NAME = 'DEPT_'||P_DEPT||'_T';
  SET STMT = 'DROP TABLE '||TABLE_NAME;
  PREPARE S1 FROM STMT;
  EXECUTE S1;
  SET STMT = 'CREATE TABLE '||TABLE_NAME||
   '( EMPNO CHAR(6) NOT NULL, '||
   'FIRSTNME VARCHAR(6) NOT NULL, '||
   'MIDINIT CHAR(1) NOT NULL, '||
   'LASTNAME CHAR(15) NOT NULL, '||
   'SALARY DECIMAL(9,2))';
  PREPARE S2 FROM STMT;
  EXECUTE S2;
  SET STMT = 'INSERT INTO '||TABLE_NAME ||
   'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY '||
   'FROM EMPLOYEE '||
   'WHERE WORKDEPT = ?';
  PREPARE S3 FROM STMT;
  EXECUTE S3 USING P_DEPT;
 END
```

Autonomous procedures

Autonomous procedures execute under their own units of work, separate from the calling program, and commit when they finish without committing the work of the calling program.

Autonomous procedures execute as separate units of work that are independent from the calling application programs. Autonomous procedures follow the rules of the COMMIT ON RETURN YES option for their changes before returning to the caller. However, their commit does not impact changes completed by the calling application program. The calling application program controls when its own updates are committed or rolled back.

If the calling application rolls back its own changes, the committed changes of the autonomous procedure are not affected. Therefore, autonomous procedures are useful for logging information about error conditions encountered by an application program. When the application encounters the error and rolls back its own changes, the committed changes of the autonomous procedure remain available.

Autonomous procedures can be called by normal application programs, other stored procedures, user-defined functions or triggers. Autonomous procedures can complete the following types of work:

- Execute SQL statements
- Invoke another procedure, function, or trigger, as long as the number of nested levels does not exceed 64, and the called procedure is not autonomous.
- Execute COMMIT and ROLLBACK statements that apply to the SQL operations executed by nested processes within the autonomous procedure.

The following restrictions apply to autonomous procedures:

- Only native SQL procedures can be defined as autonomous.
- Autonomous procedures and nested procedure, triggers, and functions within autonomous procedures cannot invoke other autonomous procedures.
- Autonomous procedures cannot see uncommitted changes from the calling application.
- When multiple versions of a procedure exist, all versions must be defined as autonomous.
- Autonomous procedures do not share locks with the calling application, meaning that the autonomous procedure might timeouts because of lock contention with the calling application.
- Parallelism is disabled for autonomous procedures. All statements in an autonomous procedure and for any nested levels within are run in sequential processing mode.
- DYNAMIC RESULT SETS 0 must be specified when autonomous procedures are used.
- Stored procedure parameters must not be defined as a LOB data type, or any distinct data type that is based on a LOB or XML value.

Related concepts

Autonomous procedures (Db2 for z/OS What's New?)

Related tasks

Controlling autonomous procedures (Db2 Administration Guide)

External stored procedures

An *external stored procedure* is a procedure that is written in a host language and can contain SQL statements. The source code for external procedures is separate from the definition.

An external stored procedure is much like any other SQL application. It can include static or dynamic SQL statements, IFI calls, and Db2 commands that are issued through IFI. You prepare external stored procedures as you would normally prepare application programs. You precompile, compile, and link-edit them. Then, you bind the DBRM into a package. You also need to define the procedure to Db2 by using the CREATE PROCEDURE statement. Thus, the source code for an external stored procedure is separate from the definition for the stored procedure.

Language requirements for the external stored procedure and its caller

You can write an external stored procedure in Assembler, C, C++, COBOL, Java, REXX, or PL/I. All programs must be designed to run using Language Environment. Your COBOL and C++ stored procedures can contain object-oriented extensions.

The program that calls the stored procedure can be in any language that supports the SQL CALL statement. ODBC applications can use an escape clause to pass a stored procedure call to Db2.

Related concepts

Object-oriented extensions in COBOL
When you use object-oriented extensions in a COBOL application, you need to consider where to place SQL statements, the SQLCA, the SQLDA, and host variable declarations. You also need to consider the rules for host variables.

REXX stored procedures
A REXX stored procedure is similar to any other REXX procedure and follows the same rules as stored procedures in other languages. A REXX stored procedure receives input parameters, executes REXX commands, optionally executes SQL statements, and returns at most one output parameter. However, a few differences exist.

Java stored procedures and user-defined functions (Db2 Application Programming for Java)

Differences between native SQL procedures and external procedures

SQL procedures are written entirely in SQL statements. External procedures are written in a host language and can contain SQL statements. You can invoke both types of procedures with an SQL CALL statement. However, you should consider several important differences in behavior and preparation.

Native SQL procedures and external procedures differ in the following ways:

How they handles errors

- For an SQL procedure, Db2 automatically returns SQL conditions in the SQLCA when the procedure does not include a RETURN statement or a handler. For information about the various ways to handle errors in an SQL procedure, see "Handling SQL conditions in an SQL procedure" on page 237.
- For an external stored procedure, Db2 does not return SQL conditions in the SQLCA to the invoking application. If you use PARAMETER STYLE SQL when you define an external procedure, you can set SQLSTATE to indicate an error before the procedure ends. For valid SQLSTATE values, see SQLSTATE values and common error codes (Db2 Codes).

How they specify the code for the stored procedure

SQL procedure definitions contain the source code for the stored procedure. An external stored procedure definition specifies the name of the stored procedure program.

How you define the stored procedure.

For both native SQL procedures and external procedures, you define the stored procedure to Db2 by executing the CREATE PROCEDURE statement. For external procedures, you must also separately bind the source code for procedure into a package. You can do this before or after you issue the CREATE PROCEDURE statement to define the external procedure.

Examples

Creating a native SQL procedure

The following example shows a definition for an SQL procedure.

```
CREATE PROCEDURE UPDATESALARY1        1
 (IN EMPNUMBR CHAR(10),               2
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL                         3
  UPDATE EMP                          4
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPNUMBR
```

Notes:

1
The stored procedure name is UPDATESALARY1.

2
The two parameters have data types of CHAR(10) and DECIMAL(6,2). Both are input parameters.

3
LANGUAGE SQL indicates that this is an SQL procedure, so a procedure body follows the other parameters.

4
The procedure body consists of a single SQL UPDATE statement, which updates rows in the employee table.

Creating an external stored procedure

The following example shows a definition for an equivalent external stored procedure that is written in COBOL. The stored procedure program, which updates employee salaries, is called UPDSAL.

```
CREATE PROCEDURE UPDATESALARY1           1
 (IN EMPNUMBR CHAR(10),                  2
 IN RATE DECIMAL(6,2))
 LANGUAGE COBOL                          3
 EXTERNAL NAME UPDSAL;                   4
```

Notes:

1
The stored procedure name is UPDATESALARY1.

2
The two parameters have data types of CHAR(10) and DECIMAL(6,2). Both are input parameters.

3
LANGUAGE COBOL indicates that this is an external procedure, so the code for the stored procedure is in a separate, COBOL program.

4
The name of the load module that contains the executable stored procedure program is UPDSAL.

Related reference

CREATE PROCEDURE (Db2 SQL)
CREATE PROCEDURE (external) (Db2 SQL)
CREATE PROCEDURE (SQL - native) (Db2 SQL)

COMMIT and ROLLBACK statements in a stored procedure

When you issue COMMIT or ROLLBACK statements in your stored procedure, Db2 commits or rolls back all changes within the unit of work. For procedures that are not defined as autonomous, the committed or rolled back changes include changes that the client application made before it called the stored procedure and Db2 work that the stored procedure does. For autonomous procedures, the committed or rolled back changes include only work done by the stored procedure's unit of work.

If your stored procedure includes COMMIT or ROLLBACK statements, define it with the one of the following clauses:

- CONTAINS SQL
- READS SQL DATA
- MODIFIES SQL DATA

The COMMIT ON RETURN clause in a stored procedure definition has no effect on the COMMIT or ROLLBACK statements in the stored procedure code. If you specify COMMIT ON RETURN YES when you define the stored procedure, Db2 issues a COMMIT statement when control returns from the stored procedure. This action occurs regardless of whether the stored procedure contains COMMIT or ROLLBACK statements.

If you specify AUTONOMOUS when you define the stored procedure, the autonomous procedure is a separate unit of work from the calling application. Db2 issues a COMMIT statement when control returns from the stored procedure, but only changes completed by the autonomous procedure are committed. Similarly, COMMIT or ROLLBACK statements in the autonomous procedure code also have no effect on work done by the calling application.

A ROLLBACK statement has the same effect on cursors in a stored procedure as it has on cursors in stand-alone programs. A ROLLBACK statement closes all open cursors. A COMMIT statement in a stored procedure closes cursors that are not declared WITH HOLD and leaves open those cursors that are declared WITH HOLD. The effect of COMMIT or ROLLBACK on cursors applies to cursors that are declared in the calling application and to cursors that are declared in the stored procedure.

Restriction: You cannot include COMMIT or ROLLBACK statements in a stored procedure if any of the following conditions are true:

- The stored procedure is nested within a trigger or user-defined function.
- The stored procedure is called by a client that uses two-phase commit processing.
- The client program uses a type 2 connection to connect to the remote server that contains the stored procedure.
- Db2 is not the commit coordinator.

If a COMMIT or ROLLBACK statement in a stored procedure violates any of these conditions, Db2 puts the transaction in a must-rollback state. Also, in this case, the CALL statement fails.

Related reference
CALL (Db2 SQL)
COMMIT (Db2 SQL)
ROLLBACK (Db2 SQL)

Special registers in a stored procedure

You can use all special registers in a stored procedure. However, you can modify only some of those special registers. After a stored procedure completes, Db2 restores all special registers to the values that they had before invocation.

Creating native SQL procedures

A *native SQL procedure* is a procedure whose body is written entirely in SQL and is created by issuing a single SQL statement, CREATE PROCEDURE.

Before you begin

Before you create a native SQL procedure, Configure Db2 for running stored procedures and user-defined functions during installation or Configure Db2 for running stored procedures and user-defined functions during migration if the native SQL procedure satisfies at least one of the following conditions:

- The native SQL procedure calls at least one external stored procedure, external SQL procedure, or user-defined function.
- The native SQL procedure is defined with ALLOW DEBUG MODE or DISALLOW DEBUG MODE. If you specify DISABLE DEBUG MODE, you do not need to set up the stored procedure environment.

About this task

A *native SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). A native SQL procedure is created by issuing a single SQL statement, CREATE PROCEDURE. Native SQL procedures do not require any other program preparation, such as precompiling, compiling, or link-editing source code. Native SQL procedures are executed as SQL statements that are bound in a Db2 package. Native SQL procedures do not have an associated external application program. Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

Procedure

To create a native SQL procedure, perform one of the following actions:

- Use IBM Data Studio to specify the source statements for the SQL procedure and deploy the SQL procedure to Db2.

  IBM Data Studio also allows you to create copies of the procedure package as needed and to deploy the procedure to remote servers.

- Manually deploy the native SQL procedure by completing the following steps:

  a) Issue the CREATE PROCEDURE statement:

  - Include a procedure body written entirely in the SQL procedural language (SQL OL). For more about what you can do within the procedure body, see *SQL-routine-body* in CREATE PROCEDURE (SQL - native) (Db2 SQL), *SQL-control-statement* in SQL procedural language (SQL PL) (Db2 SQL), and the following information:
    - "Controlling the scope of variables in an SQL procedure" on page 233
    - "Declaring cursors in an SQL procedure with nested compound statements" on page 236
    - "Handling SQL conditions in an SQL procedure" on page 237
    - "Raising a condition within an SQL procedure by using the SIGNAL or RESIGNAL statements" on page 246
  - Do not include the FENCED or EXTERNAL keywords, which specify the creation of an external SQL procedures, which are deprecated.
  - You can specify the AUTONOMOUS keyword to enable the procedure to commit without committing the work of the calling application. Autonomous procedures cannot see uncommitted changes of the calling application, and they cannot call other autonomous procedures.

  When you issue this CREATE PROCEDURE statement, the first version of this procedure is defined to Db2, and a package is implicitly bound with the options that you specify on the CREATE PROCEDURE statement.

  b) If the native SQL procedure contains one or more of the following statements or references, make copies of the native SQL procedure package, as needed:

  - CONNECT
  - SET CURRENT PACKAGESET
  - SET CURRENT PACKAGE PATH
  - A table reference with a three-part name that refers to a location other than the current server or refers to an alias that resolves to such a name.

  c) If you plan to call the native SQL procedure at another Db2 server, deploy the procedure to another Db2 for z/OS server. You can customize the bind options at the same time.

  d) Authorize the appropriate users to call the stored procedure.

What to do next

After you create a native SQL procedure, you can create additional versions of the procedure as needed. For more information, see "Creating new versions of native SQL procedures" on page 251.

Related concepts

SQL procedures
An SQL procedure is a stored procedure that contains only SQL statements.

SQL procedure body
The body of an SQL procedure contains one or more SQL statements. In the SQL procedure body, you can also declare and use variables, conditions, return codes, statements, cursors, and handlers.

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Developing database routines (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio)

Related reference

CREATE PROCEDURE (SQL - native) (Db2 SQL)

Controlling the scope of variables in an SQL procedure

Use nested compound statements within an SQL procedure to define the scope of SQL variables. You can reference the variable only within the compound statement in which it was declared and within any nested statements.

Procedure

To control the scope of a variable in an SQL procedure:

1. Declare the variable within the compound statement in which you want to reference it. Ensure that the variable name is unique within the compound statement, not including any nested statements. You can define variables with the same name in other compound statements in the same SQL procedure.
2. Reference the variable within that compound statement or any nested statements.

   Recommendation: If multiple variables with the same name exist within an SQL procedure, qualify the variable with the label from the compound statement in which it was declared. Otherwise, you might accidentally reference the wrong variable.

   If the variable name is unqualified and multiple variables with that name exist within the same scope, Db2 uses the variable in the innermost compound statement.

Example

The following example contains three declarations of the variable A. One instance is declared in the outer compound statement, which has the label OUTER1. The other instances are declared in the inner compound statements with the labels INNER1 and INNER2. In the INNER1 compound statement, Db2 presumes that the unqualified references to A in the assignment statement and UPDATE statement refer to the instance of A that is declared in the INNER1 compound statement. To refer to the instance of A that is declared in the OUTER1 compound statement, qualify the variable as OUTER1.A.

```
CREATE PROCEDURE P2 ()
         LANGUAGE SQL

 -- Outermost compound statement ------------------------
OUTER1: BEGIN  1
        DECLARE A INT DEFAULT 100;

        -- Inner compound statement with label INNER1 ---
        INNER1:  BEGIN  2
                   DECLARE A INT DEFAULT NULL;
                   DECLARE W INT DEFAULT NULL;

                   SET A = A + OUTER1.A;  3

                   UPDATE T1 SET T1.B = 5
                    WHERE T1.B = A;  4

                   SET OUTER1.A = 100;  5

                   SET INNER1.A = 200;  6
       END INNER1;  7
       -- End of inner compound statement INNER1 ------

       -- Inner compound statement with label INNER2 ---
       INNER2:  BEGIN  8
                  DECLARE A INT DEFAULT NULL;
                  DECLARE Z INT DEFAULT NULL;

                  SET A = A + OUTER1.A;

       END INNER2;  9
       -- End of inner compound statement INNER2 ------

      SET OUTER1.A = 100;  10
```

```
END OUTER1 11
```

The preceding example has the following parts:

1. The beginning of the outermost compound statement, which has the label OUTER1.
2. The beginning of the inner compound statement with the label INNER1.
3. The unqualified variable A refers to INNER1.A.
4. The unqualified variable A refers to INNER1.A.
5. OUTER1.A is a valid reference, because this variable is referenced in a nested compound statement.
6. INNER1.A is a valid reference, because this variable is referenced in the same compound statement in which it is declared. You cannot reference INNER2.A, because this variable is not in the scope of this compound statement.
7. The end of the inner compound statement with the label INNER1.
8. The beginning of the inner compound statement with the label INNER2.
9. The end of the inner compound statement with the label INNER2.
10. OUTER1.A is a valid reference, because this variable is referenced in the same compound statement in which it is declared. You cannot reference INNER1.A, because this variable is declared in a nested statement and cannot be referenced in the outer statement.
11. The end of the outermost compound statement, which has the label OUTER1.

Related concepts

Variables in SQL procedures
For data that you use only within an SQL procedure, you can declare *SQL variables* and store the values in the variables. SQL variables are similar to host variables in external stored procedures. SQL variables can be defined with the same data types and lengths as SQL procedure parameters.

References to SQL parameters and variables (Db2 SQL)

Nested compound statements in native SQL procedures
Nested compound statements are blocks of SQL statements that are contained by other blocks of SQL statements in native SQL procedures. Use nested compound statements to define condition handlers that execute more than one statement and to define different scopes for variables and condition handlers.

The following pseudo code shows a basic structure of an SQL procedure with nested compound statements:

```
CREATE PROCEDURE...
  OUTERMOST:  BEGIN
              ...
                INNER1:  BEGIN
                         ...
                           INNERMOST:  BEGIN
                                       ...
                                       ...
                                       END INNERMOST;
                         END INNER1;
                INNER2:  BEGIN
                         ...
                         ...
                         END INNER2;
             END OUTERMOST
```

In the preceding code, the OUTERMOST compound statement contains two nested compound statements: INNER1 and INNER2. INNER1 contains one nested compound statement: INNERMOST.

Related concepts

Handlers in an SQL procedure

If an error occurs when an SQL procedure executes, the procedure ends unless you include statements to tell the procedure to perform some other action. These statements are called handlers.

Related tasks

Defining condition handlers that execute more than one statement
A *condition handler* defines the action that an SQL procedure takes when a particular condition occurs. You must specify the action as a single SQL procedure statement.

Statement labels for nested compound statements in native SQL procedures
You can define a label for each compound statement in an SQL procedure. This label enables you to reference this block of statements in other statements such as the GOTO, LEAVE, and ITERATE SQL PL control statements. You can also use the label to qualify a variable when necessary. Labels are not required.

A label name must meet the following criteria:

- Be unique within the compound statement, including any compound statements that are nested within the compound statement.
- Not be the same as the name of the SQL procedure.

You can reference a label within the compound statement in which it is defined, including any compound statements that are nested within that compound statement.

Example of statement labels: The following example shows several statement labels and their scope:

```
CREATE PROCEDURE P1 ()
         LANGUAGE SQL

 --Outermost compound statement ------------------------
OUTER1: BEGIN  1

        --Inner compound statement with label INNER1 ---
        INNER1: BEGIN  2
          IF...
            ABC:  LEAVE INNER1;  3
          ELSEIF
            XYZ:  LEAVE OUTER1;  4
          END IF

        END INNER1;
        --End of inner compound statement INNER1 ------

        --Inner compound statement with label INNER2---
        INNER2: BEGIN  5
          XYZ:...statement  6
        END INNER2;
        -- End of inner compound statement INNER2 -----

END OUTER1  7
```

The preceding example has the following parts:

1. The beginning of the outermost compound statement, which is labeled OUTER1
2. The beginning of an inner compound statement that is labeled INNER1
3. A LEAVE statement that is defined with the label ABC. This LEAVE statement specifies that Db2 is to terminate processing of the compound statement INNER1 and begin processing the next statement, which is INNER2. This LEAVE statement cannot specify INNER2, because that label is not within the scope of the INNER1 compound statement.
4. A LEAVE statement that is defined with the label XYZ. This LEAVE statement specifies that Db2 is to terminate processing of the compound statement OUTER1 and begin processing the next statement, if one exists. This example does not show the next statement.
5. The beginning of an inner compound statement that is labeled INNER2.
6. A statement that is defined with the label XYZ. This label is acceptable even though another statement in this procedure has the same label, because the two labels are in different scopes. Neither label is contained within the scope of the other.

7. The end of the outermost compound statement that is labeled OUTER1.

The following examples show valid and invalid uses of labels:

Invalid example of labels:

```
L1:  BEGIN
     L2: SET A = B;
     L1: GOTO L2:  --This duplicate use of the label L1 causes an error, because
                   --the same label is already used in the same scope.

END L1;
```

Valid example of labels:

```
L1:  BEGIN
  L2:  BEGIN
    L4: BEGIN  --This line contains the first use of the label L4
      DECLARE A CHAR(5);
      SET A = B;
    END L4;
  END L2;

  L3:  BEGIN
    L4:  BEGIN  --This second use of the label L4 is valid, because
                --it is used in a different scope.
      DECLARE A CHAR(5);
      SET A = B;
    END L4;
  END L3;
END L1;
```

Declaring cursors in an SQL procedure with nested compound statements

When you declare a cursor in an SQL procedure that has nested compound statements, you cannot necessarily reference the cursor anywhere in the procedure. The scope of the cursor is constrained to the compound statement in which you declare it.

Procedure

Specify the DECLARE CURSOR statement within the compound statement in which you want to reference the cursor. Use a cursor name that is unique within the SQL procedure.

You can reference the cursor within the compound statement in which it is declared and within any nested statements. If the cursor is declared as a result set cursor, even if the cursor is not declared in the outermost compound statement, any calling application can reference it.

Example

In the following example, cursor X is declared in the outer compound statement. This cursor can be referenced within the outer block in which it was declared and within any nested compound statements.

```
CREATE PROCEDURE SINGLE_CSR
   (INOUT IR1 INT, INOUT JR1 INT, INOUT IR2 INT, INOUT JR2 INT)
   LANGUAGE SQL
   DYNAMIC RESULT SETS 2
  BEGIN
    DECLARE I INT;
    DECLARE J INT;
    DECLARE X CURSOR WITH RETURN FOR --outer declaration for X
       SELECT * FROM CSRT1;

   SUB: BEGIN
          OPEN X;                    --references X in outer block
          FETCH X INTO I,J;          --references X in outer block
          SET IR1 = I;
          SET JR1 = J;
        END;

  FETCH X INTO I,J;                --references X in outer block
  SET IR2 = I;
  SET JR2 = j;
```

```
  CLOSE X;
 END
```

Related reference

CREATE PROCEDURE (SQL - native) (Db2 SQL)

DECLARE CURSOR (Db2 SQL)

Handling SQL conditions in an SQL procedure

In an SQL procedure, you can specify how the program should handle certain SQL errors and SQL warnings.

About this task

If you do not include a handler or a RETURN statement in the SQL procedure, Db2 automatically returns any SQL conditions to the caller in the SQLCA.

Procedure

To handle SQL conditions, use one of the following techniques:

- Include statements called *handlers* to tell the procedure to perform some other action when an error or warning occurs.
- Include a RETURN statement in an SQL procedure to return an integer status value to the caller.
- Include a SIGNAL statement or a RESIGNAL statement to raise a specific SQLSTATE and to define the message text for that SQLSTATE.
- Force a negative SQLCODE to be returned by a procedure if a trigger calls the procedure.

Handlers in an SQL procedure

If an error occurs when an SQL procedure executes, the procedure ends unless you include statements to tell the procedure to perform some other action. These statements are called handlers.

Handlers are similar to WHENEVER statements in external SQL application programs. Handlers tell the SQL procedure what to do when an error or warning occurs, or when no more rows are returned from a query. In addition, you can declare handlers for specific SQLSTATEs. You can refer to an SQLSTATE by its number in a handler, or you can declare a name for the SQLSTATE and then use that name in the handler.

The general form of a handler declaration is:

```
DECLARE handler-type HANDLER FOR condition SQL-procedure-statement;
```

In general, the way that a handler works is that when an error occurs that matches *condition*, the *SQL-procedure-statement* executes. When the *SQL-procedure-statement* completes, Db2 performs the action that is indicated by *handler-type*.

Types of handlers

The handler type determines what happens after the completion of the *SQL-procedure-statement*. You can declare the handler type to be either CONTINUE or EXIT:

CONTINUE
: Specifies that after *SQL-procedure-statement* completes, execution continues with the statement after the statement that caused the error.

EXIT
: Specifies that after *SQL-procedure-statement* completes, execution continues at the end of the compound statement that contains the handler.

Example: CONTINUE handler: This handler sets flag at_end when no more rows satisfy a query. The handler then causes execution to continue after the statement that returned no rows.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET at_end=1;
```

Example: EXIT handler: This handler places the string 'Table does not exist' into output parameter OUT_BUFFER when condition NO_TABLE occurs. NO_TABLE is previously declared as SQLSTATE 42704 (*name* is an undefined name). The handler then causes the SQL procedure to exit the compound statement in which the handler is declared.

```
DECLARE NO_TABLE CONDITION FOR '42704';
⋮
DECLARE EXIT HANDLER FOR NO_TABLE
 SET OUT_BUFFER='Table does not exist';
```

Defining condition handlers that execute more than one statement
A *condition handler* defines the action that an SQL procedure takes when a particular condition occurs. You must specify the action as a single SQL procedure statement.

Procedure

To define a condition handler that executes more than one statement when the specified condition occurs, specify a compound statement within the declaration of that handler.

Examples

Example
The following example shows a condition handler that captures the SQLSTATE value and sets a local flag to TRUE.

```
BEGIN
   DECLARE SQLSTATE CHAR(5);
   DECLARE PrvSQLState CHAR(5) DEFAULT '00000';
   DECLARE ExceptState INT;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
           SET PrvSQLState = SQLSTATE;
           SET ExceptState = TRUE;
      END;
   ...
END
```

Example
The following example declares a condition handler for SQLSTATE 72822. The subsequent SIGNAL statement is within the scope of this condition handler and thus activates this handler. The condition handler tests the value of the SQL variable VAR with an IF statement. Depending on the value of VAR, the SQLSTATE is changed and the message text is set.

```
DECLARE EXIT HANDLER FOR SQLSTATE '72822'
       IF ( VAR = 'OK' ) THEN
         RESIGNAL SQLSTATE '72623'
                  SET MESSAGE_TEXT = 'Got SQLSTATE 72822';
       ELSE
         RESIGNAL SQLSTATE '72319'
                  SET MESSAGE_TEXT = VAR;
       END IF;

SIGNAL SQLSTATE '72822';
```

Related reference
compound-statement (Db2 SQL)

Controlling how errors are handled within different scopes in an SQL procedure
You can use nested compound statements in an SQL procedure to specify that errors be handled differently within different scopes. You can also ensure that condition handlers are checked only with a particular compound statement.

Procedure

To control how errors are handled within different scopes in an SQL procedure:

1. Optional: Declare a condition by specifying a DECLARE CONDITION statement within the compound statement in which you want to reference it. You can reference a condition in the declaration of a condition handler, a SIGNAL statement, or a RESIGNAL statement.

   Restriction: If multiple conditions with that name exist within the same scope, you cannot explicitly refer to a condition that is not the most local in scope. Db2 uses the condition in the innermost compound statement.
2. Declare a condition handler by specifying a DECLARE HANDLER statement within the compound statement to which you want the condition handler to apply. Within the declaration of the condition handler, you can specify a previously defined condition.

   Restriction: Condition handlers that are declared in the same compound statement cannot handle conditions encountered in each other or themselves.

Examples

Example

In the following example, a condition with the name ABC is declared twice, and a condition named XYZ is declared once.

```
CREATE PROCEDURE...
       DECLARE ABC CONDITION...

       DECLARE XYZ CONDITION...
        BEGIN
          DECLARE ABC CONDITION...
          SIGNAL ABC;  1
        END;

       SIGNAL ABC;  2
```

The following notes refer to the preceding example:

1. ABC refers to the condition that is declared in the innermost block. If this statement were changed to SIGNAL XYZ, XYZ would refer to the XYZ condition that is declared in the outermost block.
2. ABC refers to the condition that is declared in the outermost block.

Example

The following example contains multiple declarations of a condition with the name FOO, and a single declaration of a condition with the name GORP.

```
CREATE PROCEDURE MYTEST (INOUT A CHAR(1), INOUT B CHAR(1))
  L1: BEGIN
    DECLARE GORP CONDITION
        FOR SQLSTATE '33333'; -- defines a condition with the name GORP for SQLSTATE 33333

    DECLARE EXIT HANDLER FOR GORP --defines a condition handler for SQLSTATE 33333
      L2: BEGIN
        DECLARE FOO CONDITION
            FOR SQLSTATE '12345'; --defines a condition with the name FOO for SQLSTATE 12345
        DECLARE CONTINUE HANDLER FOR FOO --defines a condition handler for SQLSTATE 12345
          L3: BEGIN
           SET A = 'A';
           ...more statements...
          END L3;
        SET B = 'B';

        IF...
          SIGNAL FOO; --raises SQLSTATE 12345
        ELSEIF
          SIGNAL GORP; --raises SQLSTATE 33333
        END IF;

     END L2;

   L4: BEGIN
     DECLARE FOO CONDITION
         FOR SQLSTATE '54321' --defines a condition with the name FOO for SQLSTATE 54321
     DECLARE EXIT HANDLER FOR FOO...; --defines a condition handler for SQLSTATE 54321

     SIGNAL FOO SET MESSAGE_TEXT = '...'; --raises SQLSTATE 54321
```

```
     L5: BEGIN
       DECLARE FOO CONDITION
           FOR SQLSTATE '99999'; --defines a condition with the name FOO for SQLSTATE 99999
       ...more statements...
     END L5;

   END L4;

   --At this point, the procedure cannot reference FOO, because this condition is not defined
   --in this outer scope

END L1
```

Example

In the following example, the compound statement with the label OUTER contains two other compound statements: INNER1A and INNER1B. The INNER1A compound statement contains another compound statement, which has the label INNER1A2, and the declaration for a condition handler HINNER1A. The body of the condition handler HINNER1A contains another compound statement, which defines another condition handler, HINNER1A_HANDLER.

```
OUTER:
  BEGIN                                  <=============.
    -- Handler for OUTER                               |
    DECLARE ... HANDLER  -- HOUTER                     |
      BEGIN                              <---.         |
        :                                    |         |
      END; -- End of handler             <---.         |
    :                                                  |
    :                                                  |
                                                       |
    -- Level 1 - first compound statement              |
    INNER1A:                                           |
      BEGIN                              <---------.   |
        -- Handler for INNER1A                     |   |
        DECLARE ... HANDLER -- HINNER1A            |   |
          BEGIN                          <------.  |   |
            -- Handler for handler HINNER1A     |  |   |
            DECLARE...HANDLER --HINNER1A_HANDLER|  |   |
              BEGIN                      <---.  |  |   |
                :                            |  |  |   |
              END; -- End of handler     <---.  |  |   |
            :                                   |  |   |
            : -- stmt that gets condition       |  |   |  2
            :                                   |  |   |
            : -- more statements in handler     |  |   |
          END; -- End of HINNER1A handler<------.  |   |
                                                   |   |
        INNER1A2:                                  |   |
          BEGIN                              <--.  |   |
            DECLARE ... HANDLER...-- HINNER1A2  |  |   |
              BEGIN;                     <---.  |  |   |
                :                            |  |  |   |
              END; -- End of handler     <---.  |  |   |
            :                                   |  |   |
            : -- statement that gets condition  |  |   |  1
            :                                   |  |   |
            : -- statement after statement      |  |   |
            :    -- that encountered condition  |  |   |
          END INNER1A2;                      <--'  |   |
        :                                          |   |
        : -- statements in INNER1A                 |   |
      END INNER1A;                       <---------'   |
                                                       |
    -- Level 1 - second compound statement             |
    INNER1B:                                           |
      BEGIN                              <---------.   |
        -- Handler for handler INNER1B             |   |
        DECLARE ...HANDLER -- HINNER1B             |   |
          BEGIN                           <------. |   |
            -- Handler for HINNER1B --           | |   |
            DECLARE ...HANDLER --HINNER1B_HANDLER| |   |
              BEGIN                      <---.   | |   |
                :                            |   | |   |
              END; -- End of handler     <---.   | |   |
            :                                    | |   |
            : -- statements in handler           | |   |
          END; -- End of HINNER1B handler<-------. |   |
      :                                            |   |
      : -- statements in INNER1B                   |   |
```

```
      END INNER1B;                         <---------'   |
                                                         |
      :  -- statements in OUTER                          |
    END OUTER;                             <============='
```

The following notes apply to the preceding example:

1. If an exception, warning, or NOT FOUND condition occurs within the INNER1A2 compound statement, the most appropriate handler within that compound statement is activated to handle the condition. Then, one of the following actions occurs depending on the type of condition handler:

   - If the condition handler (HINNER1A2) is an exit handler, control is returned to the end of the compound statement that contained the condition handler.
   - If the condition handler (HINNER1A2) is a continue handler, processing continues with the statement after the statement that encountered the condition.

   If no appropriate handler exists in the INNER1A2 compound statement, Db2 considers the following handlers in the specified order:

   a. The most appropriate handler within the INNER1A compound statement.
   b. The most appropriate handler within the OUTER compound statement.

   If no appropriate handler exists in the OUTER compound statement, the condition is an unhandled condition. If the condition is an exception condition, the procedure terminates and returns an unhandled condition to the invoking application. If the condition is a warning or NOT FOUND condition, the procedure returns the unhandled warning condition to the invoking application.

2. If an exception, warning, or NOT FOUND condition occurs within the body of the condition handler HINNER1A, and the condition handler HINNER1A_HANDLER is the most appropriate handler for the exception, that handler is activated. Otherwise, the most appropriate handler within the OUTER compound statement handles the condition. If no appropriate handler exists within the OUTER compound statement, the condition is treated as an unhandled condition.

Example

In the following example, when *statement2* results in a NOT FOUND condition, the appropriate condition handler is activated to handle the condition. When the condition handler completes, the compound statement that contains that condition handler terminates, because the condition handler is an EXIT handler. Processing then continues with *statement4*.

```
BEGIN
  DECLARE EXIT HANDLER FOR NOT FOUND
    SET OUT_OF_DATA_FLAG = ON;
  statement1...
  statement2... --assume that this statement results in a NOT FOUND condition
  statement3...
END;

statement4
...
```

Example

In the following example, Db2 checks for SQLSTATE 22H11 only for statements inside the INNER compound statement. Db2 checks for SQLEXCEPTION for all statements in both the OUTER and INNER blocks.

```
OUTER: BEGIN
         DECLARE var1 INT;
         DECLARE EXIT HANDLER FOR SQLEXCEPTION
            RETURN -3;

         INNER: BEGIN
                 DECLARE EXIT HANDLER FOR SQLSTATE '22H11'
                    RETURN -1;
                 DECLARE C1 CURSOR FOR SELECT col1 FROM table1;
                 OPEN C1;
                 CLOSE C1;
                   :
                   :  -- more statements
```

```
        END INNER;
    :
    :  -- more statements
```

Example

In the following example, Db2 checks for SQLSTATE 42704 only for statements inside the A compound statement.

```
CREATE PROCEDURE EXIT_TEST ()
   LANGUAGE SQL
   BEGIN
      DECLARE OUT_BUFFER VARCHAR(80);
      DECLARE NO_TABLE CONDITION FOR SQLSTATE '42704';

      A: BEGIN                                               1
            DECLARE EXIT HANDLER FOR NO_TABLE                3
               BEGIN
                 SET OUT_BUFFER ='Table does not exist';     4
               END;

            -- Drop potentially nonexistent table:
            DROP TABLE JAVELIN;                              2

         B: SET OUT_BUFFER ='Table dropped successfully';
      END;
      -- Copy OUT_BUFFER to some message table:
      C: INSERT INTO MESSAGES VALUES (OUT_BUFFER);           5
```

The following notes describe a possible flow for the preceding example:

1. A nested compound statement with label A confines the scope of the NO_TABLE exit handler to the statements that are specified in the A compound statement.
2. If the table JAVELIN does not exist, the DROP statement raises the NO_TABLE condition.
3. The exit handler for NO_TABLE is activated.
4. The variable OUT_BUFFER is set to the string 'Table does not exist.'
5. Execution continues with the INSERT statement. No more statements in the A compound statement are processed.

Example

The following example illustrates the scope of different condition handlers.

```
CREATE PROCEDURE ERROR_HANDLERS(IN PARAM INTEGER)
       LANGUAGE SQL
 OUTER:  BEGIN
    DECLARE I INTEGER;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';

    DECLARE EXIT HANDLER FOR
            SQLSTATE VALUE '38H02',
            SQLSTATE VALUE '38H04',
            SQLSTATE VALUE '38HI4',
            SQLSTATE VALUE '38H06'
        OUTER_HANDLER: BEGIN
           DECLARE TEXT VARCHAR(70);
           SET TEXT = SQLSTATE ||
                      ' RECEIVED AND MANAGED BY OUTER ERROR HANDLER' ;
           RESIGNAL SQLSTATE VALUE '38HE0'
                    SET MESSAGE_TEXT = TEXT;
        END OUTER_HANDLER;

    INNER: BEGIN
      DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H03'
         RESIGNAL SQLSTATE VALUE '38HI3'
            SET MESSAGE_TEXT = '38H03 MANAGED BY INNER ERROR HANDLER';

      DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H04'
         RESIGNAL SQLSTATE VALUE '38HI4'
            SET MESSAGE_TEXT = '38H04 MANAGED BY INNER ERROR HANDLER';

      DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H05'
         RESIGNAL SQLSTATE VALUE '38HI5'
            SET MESSAGE_TEXT = '38H05 MANAGED BY INNER ERROR HANDLER';
```

```
      CASE PARAM
        WHEN 1 THEN                              -- (1)
          SIGNAL SQLSTATE VALUE '38H01'
            SET MESSAGE_TEXT =
                'EXAMPLE 1: ERROR SIGNALED FROM INNER COMPOUND STMT';

        WHEN 2 THEN                              -- (2)
          SIGNAL SQLSTATE VALUE '38H02'
            SET MESSAGE_TEXT =
                'EXAMPLE 2: ERROR SIGNALED FROM INNER COMPOUND STMT';

        WHEN 3 THEN                              -- (3)
          SIGNAL SQLSTATE VALUE '38H03'
            SET MESSAGE_TEXT =
                'EXAMPLE 3: ERROR SIGNALED FROM INNER COMPOUND STMT';

        WHEN 4 THEN                              -- (4)
          SIGNAL SQLSTATE VALUE '38H04'
            SET MESSAGE_TEXT =
                'EXAMPLE 4: ERROR SIGNALED FROM INNER COMPOUND STMT';

        ELSE
            SET I = 1; /*Do not do anything */
     END CASE;
  END INNER;

  CASE PARAM
    WHEN 5 THEN                                  -- (5)
       SIGNAL SQLSTATE VALUE '38H05'
         SET MESSAGE_TEXT =
              'EXAMPLE 5: ERROR SIGNALED FROM OUTER COMPOUND STMT';
     WHEN 6 THEN                                  -- (6)
        SIGNAL SQLSTATE VALUE '38H06'
          SET MESSAGE_TEXT =
              'EXAMPLE 6: ERROR SIGNALED FROM OUTER COMPOUND STMT';
     ELSE                                         -- (7)
        SET I = 1; /*Do not do anything */
   END CASE;
 END OUTER
```

The following table summarizes the behavior of the preceding example:

Input value for PARM	Expected behavior
1	SQLSTATE 38H01 is signaled from the INNER compound statement. Because no appropriate handler exists, the procedure terminates and returns the unhandled exception condition, 38H01 with SQLCODE -438, to the calling application.
2	SQLSTATE 38H02 is signaled from the INNER compound statement. The condition handler in the OUTER compound statement is activated. A RESIGNAL statement, with SQLSTATE 38HE0, is issued from within the body of the condition handler. This exception causes control to be returned to the end of the OUTER compound statement with exception condition 38HE0 and SQLCODE -438. The procedure terminates and returns the unhandled condition to the calling application.
3	SQLSTATE 38H03 is signaled from the INNER compound statement. A condition handler within the INNER compound statement is activated. A RESIGNAL statement, with SQLSTATE 38HI3, is issued from within the body of the condition handler. Because no appropriate handler exists, the procedure terminates and returns the unhandled exception condition, 38HI3 with SQLCODE -438, to the calling application.

Input value for PARM	Expected behavior
4	SQLSTATE 38H04 is signaled from the INNER compound statement. A condition handler within the INNER compound statement is activated. A RESIGNAL statement, with SQLSTATE 38HI4, is issued from within the body of the condition handler. A condition handler in the OUTER compound statement is activated. A RESIGNAL statement, with SQLSTATE 38HE0, is issued from within the body of the condition handler. This exception causes control to be returned to the end of the OUTER compound statement with exception condition 38HE0 and SQLCODE -438. The procedure terminates and returns the unhandled condition to the calling application.
5	SQLSTATE 38H05 is signaled from the OUTER compound statement. Because no appropriate handler exists, the procedure terminates and returns the unhandled exception condition, 38H05 with SQLCODE -438, to the calling application.
6	SQLSTATE 38H06 is signaled from the OUTER compound statement. A condition handler in the OUTER compound statement is activated. A RESIGNAL statement, with SQLSTATE 38HE0, is issued from within the body of the condition handler. This exception causes control to be returned to the end of the OUTER compound statement with exception condition 38HE0 and SQLCODE -438. The procedure terminates and returns the unhandled condition to the calling application.
7	The ELSE clause of the CASE statement executes and processes the SET statement. A successful completion code is returned to the calling application.

Example

In the following example SQL procedure, the condition handler for exception1 is not within the scope of the condition handler for exception0. If exception condition exception1 is raised in the body of the condition handler for exception0, no appropriate handler exists, and the procedure terminates with an unhandled exception.

```
CREATE PROCEDURE divide (  .....)
 LANGUAGE SQL CONTAINS SQL
 BEGIN
    DECLARE dn_too_long CHAR(5) DEFAULT 'abcde';

    -- Declare condition names -------------------------
    DECLARE exception0  CONDITION FOR SQLSTATE '22001';
    DECLARE exception1  CONDITION FOR SQLSTATE 'xxxxx';

    -- Declare cursors ---------------------------------
    DECLARE cursor1 CURSOR WITH RETURN FOR
      SELECT * FROM dept;

    -- Declare handlers --------------------------------
    DECLARE CONTINUE HANDLER FOR exception0
       BEGIN
          some SQL statement that causes an error 'xxxxx'
       END

    DECLARE CONTINUE HANDLER FOR exception1
       BEGIN
          ...
       END


    -- Mainline of procedure ---------------------------
    INSERT INTO  DEPT (DEPTNO) VALUES (dn_too_long);
       -- Assume that this statement results in SQLSTATE '22001'

    OPEN CURSOR1;
END
```

Retrieving diagnostic information by using GET DIAGNOSTICS in a handler
Handlers specify the action that an SQL procedure takes when a particular error or condition occurs. In some cases, you want to retrieve additional diagnostic information about the error or warning condition.

About this task

Procedure

You can include a GET DIAGNOSTICS statement in a handler to retrieve error or warning information.

If you include GET DIAGNOSTICS, it must be the first statement that is specified in the handler.

Example: Using GET DIAGNOSTICS to retrieve message text

Suppose that you create an SQL procedure, named divide1, that computes the result of the division of two integers. You include GET DIAGNOSTICS to return the text of the division error message as an output parameter:

```
CREATE PROCEDURE divide1
  (IN numerator INTEGER, IN denominator INTEGER,
   OUT divide_result INTEGER, OUT divide_error VARCHAR(1000))
  LANGUAGE SQL
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      GET DIAGNOSTICS CONDITION 1 divide_error = MESSAGE_TEXT;
    SET divide_result = numerator / denominator;
  END
```

Ignoring a condition in an SQL procedure
You can specify that you want to ignore errors or warnings within a particular scope of statements in an SQL procedure. However, do so with caution.

Procedure

Declare a condition handler that contains an empty compound statement.

Example

The following example shows a condition handler that is declared as a way of ignoring a condition. Assume that your SQL procedure inserts rows into a table that has a unique column. If the value to be inserted for that column already exists in the table, the row is not inserted. However, in this case, you do not want Db2 to notify the application about this condition, which is indicated by SQLSTATE 23505.

```
      DECLARE CONTINUE HANDLER FOR SQLSTATE '23505'
        BEGIN  -- ignore error for duplicate value
        END;
```

Related concepts

Handlers in an SQL procedure
If an error occurs when an SQL procedure executes, the procedure ends unless you include statements to tell the procedure to perform some other action. These statements are called handlers.

Related reference

SQLSTATE values and common error codes (Db2 Codes)

Raising a condition within an SQL procedure by using the SIGNAL or RESIGNAL statements

Within an SQL procedure, you can force a particular condition to occur with a specific SQLSTATE and message text.

About this task

You can use either a SIGNAL or RESIGNAL statement to raise a condition with a specific SQLSTATE and message text within an SQL procedure. The SIGNAL and RESIGNAL statements differ in the following ways:

- You can use the SIGNAL statement anywhere within an SQL procedure. You must specify the SQLSTATE value. In addition, you can use the SIGNAL statement in a trigger body. For information about using the SIGNAL statement in a trigger, see "Creating a trigger" on page 151.
- You can use the RESIGNAL statement only within a handler of an SQL procedure. If you do not specify the SQLSTATE value, Db2 uses the same SQLSTATE value that activated the handler.

You can use any valid SQLSTATE value in a SIGNAL or RESIGNAL statement, except an SQLSTATE class with '00' as the first two characters.

The following table summarizes the differences between issuing a RESIGNAL or SIGNAL statement within the body of a condition handler. For each row in the table, assume that the diagnostics area contains the following information when the RESIGNAL or SIGNAL statement is issued:

```
RETURNED_SQLSTATE    xxxxx
MESSAGE_TEXT 'this is my message'
```

Table 50. Example RESIGNAL and SIGNAL statements

Specify a new condition?	Specify message text?	Example RESIGNAL statement...	Example SIGNAL statement...	Result
No	No	`RESIGNAL` 1	Not possible	RETURNED_SQLSTATE xxxxx MESSAGE_TEXT 'this is my message'
Yes	No	`RESIGNAL '98765'` 2	`SIGNAL '98765'`	RETURNED_SQLSTATE 98765 MESSAGE_TEXT 'APPLICATION RAISED ERROR WITH DIAGNOSTIC TEXT: this is my message'
No	Yes	Not possible	Not possible	NA
Yes	Yes	`RESIGNAL '98765' SET MESSAGE_TEXT = 'xyz'` 3	`SIGNAL '98765' SET MESSAGE_TEXT = 'xyz'` 3	RETURNED_SQLSTATE 98765 MESSAGE_TEXT 'APPLICATION RAISED ERROR WITH DIAGNOSTIC TEXT: xyz'

Note:

1. This statement raises the current condition with the existing SQLSTATE, SQLCODE, message text, and tokens.
2. This statement raises a new condition (SQLSTATE '98765'). Existing message text and tokens are reset. The SQLCODE is set to -438 for an error or 438 for a warning.

3. This statement raises a new condition (SQLSTATE '98765') with new message text ('xyz'). The SQLCODE is set to -438 for an error or 438 for a warning.

Example of the SIGNAL statement in an SQL procedure
You can use the SIGNAL statement anywhere within an SQL procedure to raise a particular condition.

The following example uses an ORDERS table and a CUSTOMERS table that are defined in the following way:

```
CREATE TABLE ORDERS
  (ORDERNO     INTEGER NOT NULL,
   PARTNO      INTEGER NOT NULL,
   ORDER_DATE  DATE DEFAULT,
   CUSTNO      INTEGER NOT NULL,
   QUANTITY    SMALLINT NOT NULL,
   CONSTRAINT REF_CUSTNO FOREIGN KEY (CUSTNO)
     REFERENCES CUSTOMERS (CUSTNO) ON DELETE RESTRICT,
   PRIMARY KEY (ORDERNO,PARTNO));
```

```
CREATE TABLE CUSTOMERS
  (CUSTNO      INTEGER NOT NULL,
   CUSTNAME    VARCHAR(30),
   CUSTADDR    VARCHAR(80),
   PRIMARY KEY (CUSTNO));
```

Example: Using SIGNAL to set message text

Suppose that you have an SQL procedure for an order system that signals an application error when a customer number is not known to the application. The ORDERS table has a foreign key to the CUSTOMERS table, which requires that the CUSTNO exist in the CUSTOMERS table before an order can be inserted:

```
CREATE PROCEDURE submit_order
  (IN ONUM INTEGER, IN PNUM INTEGER,
   IN CNUM INTEGER, IN QNUM INTEGER)
  LANGUAGE SQL
  MODIFIES SQL DATA
  BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE VALUE '23503'
      SIGNAL SQLSTATE '75002'
        SET MESSAGE_TEXT = 'Customer number is not known';
    INSERT INTO ORDERS (ORDERNO, PARTNO, CUSTNO, QUANTITY)
                VALUES (ONUM, PNUM, CNUM, QNUM);
  END
```

In this example, the SIGNAL statement is in the handler. However, you can use the SIGNAL statement to invoke a handler when a condition occurs that will result in an error.

Related concepts

Example of the RESIGNAL statement in a handler
You can use the RESIGNAL statement in an SQL procedure to assign a different value to the condition that activated the handler. T

Example of the RESIGNAL statement in a handler
You can use the RESIGNAL statement in an SQL procedure to assign a different value to the condition that activated the handler. T

Example: Using RESIGNAL to set an SQLSTATE valu

Suppose that you create an SQL procedure, named divide2, that computes the result of the division of two integers. You include SIGNAL to invoke the handler with an overflow condition that is caused by a zero divisor, and you include RESIGNAL to set a different SQLSTATE value for that overflow condition:

```
CREATE PROCEDURE divide2
  (IN numerator INTEGER, IN denominator INTEGER,
   OUT divide_result INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE overflow CONDITION FOR SQLSTATE '22003';
```

```
    DECLARE CONTINUE HANDLER FOR overflow
      RESIGNAL SQLSTATE '22375';
    IF denominator = 0 THEN
      SIGNAL overflow;
    ELSE
      SET divide_result = numerator / denominator;
    END IF;
  END
```

Example: RESIGNAL in a nested compound statement

If the following SQL procedure is invoked with argument values 1, 0, and 0, the procedure returns a value of 2 for RC and sets the oparm1 parameter to 650.

```
CREATE PROCEDURE resig4
    (IN iparm1 INTEGER, INOUT oparm1 INTEGER, INOUT rc INTEGER)
 LANGUAGE SQL
  A1: BEGIN
       DECLARE c1 INT DEFAULT 1;
       DECLARE CONTINUE HANDLER FOR  SQLSTATE VALUE '01ABX'
         BEGIN
          .... some other statements
          SET RC = 3;                                      6
         END;

       A2: SET oparm1 = 5;                                 1

       A3: BEGIN
             DECLARE c1 INT DEFAULT 1;
             DECLARE CONTINUE HANDLER
                 FOR SQLSTATE VALUE '01ABC'
               BEGIN
                 SET RC = 1;                               4
                 RESIGNAL SQLSTATE VALUE '01ABX'           5
                   SET MESSAGE_TEXT = 'get out of here';
                 SET RC = 2;                                7
               END;

             A7: SET oparm1 = oparm1 + 110;                2
             SIGNAL SQLSTATE VALUE '01ABC'                 3
                SET MESSAGE_TEXT = 'yikes';
             SET oparm1 = oparm1 + 215;                     8
           END;
        SET oparm1 = oparm1 + 320;                          9

END
```

The following notes refer to the preceding example:

1. oparm1 is initially set to 5.
2. oparm1 is incremented by 110. The value of oparm1 is now 115.
3. The SIGNAL statement causes the condition handler that is contained in the A3 compound statement to be activated.
4. In this condition handler, RC is set to 1.
5. The RESIGNAL statement changes the SQLSTATE to 01ABX. This value causes the continue handler in the A1 compound statement to be activated.
6. RC is set to 3 in this condition handler. Because this condition handler is a continue handler, when the handler action completes, control returns to the SET statement after the RESIGNAL statement.
7. RC is set to 2 in this condition handler. Because this condition handler is a continue handler, control returns to the SET statement that follows the SIGNAL statement that caused the condition handler to be activated.
8. oparm1 is incremented by 215. The value of oparm is now 330.
9. oparm1 is incremented by 320. The value of oparm is now 650.

How SIGNAL and RESIGNAL statements affect the diagnostics area
When you issue a SIGNAL statement, a new logical diagnostics area is created. When you issue a RESIGNAL statement, the current diagnostics area is updated.

When you issue a SIGNAL statement, a new diagnostics area is logically created. In that diagnostics area, RETURNED_SQLSTATE is set to the SQLSTATE or condition name specified. If you specified message text as part of the SIGNAL statement, MESSAGE_TEXT in the diagnostics area is also set to the specified value.

When you issue a RESIGNAL statement with a SQLSTATE value, condition name, or message text, the current diagnostics area is updated with the specified information.

Making copies of a package for a native SQL procedure

When you create a native SQL procedure, a package is implicitly bound with the options that you specified on the CREATE PROCEDURE statement. If the native SQL procedure performs certain actions, you need to explicitly make copies of that package.

About this task

If the native SQL procedure performs one or more of the following actions, you need to create copies of the package for that procedure:

- Uses a CONNECT statement to connect to a database server.
- Refers to a table with a three part name that includes a location other than the current server or refers to an alias that resolves to such a name.
- Sets the CURRENT PACKAGESET special register to control which package is invoked for that version of the procedure.
- Sets the CURRENT PACKAGE PATH special register to control which package is invoked for that version of the procedure.

The package for a version of a procedure has the following name: *location*.*collection-id*.*package-id*.*version-id* where these variables have the following values:

location
Value of the CURRENT SERVER special register

collection-id
Schema qualifier of the procedure

package-id
Procedure name

version-id
Version identifier

To make copies of a package for a native SQL procedure, specify the BIND PACKAGE command with the COPY option. For copies that are created on the current server, specify a different schema qualifier, which is the collection ID. For the first copy that is created on a remote server, you can specify the same schema qualifier. For other copies on that remote server, specify a different schema qualifier.

If you later change the native SQL procedure, you might need to explicitly rebind any local or remote copies of the package that exist for that version of the procedure.

Examples

Example
Because the following native SQL procedure contains a CONNECT statement, you must create a copy of the package at the target server, which in this case is at location SAN_JOSE. The subsequent BIND command creates a copy of the package for version ABC of the procedure TEST.MYPROC. This package is created at location SAN_JOSE and is used by Db2 when this procedure is executed.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
 ...
 CONNECT TO SAN_JOSE
```

```
  ...
END

BIND PACKAGE (SAN_JOSE.TEST) COPY(TEST.MYPROC) COPYVER(ABC) ACTION(ADD)
```

Example

The following native SQL procedure sets the CURRENT PACKAGESET special register to ensure that Db2 uses the package with the collection ID COLL2 for this version of the procedure. Consequently, you must create such a package. The subsequent BIND command creates this package with collection ID COLL2. This package is a copy of the package for version ABC of the procedure TEST.MYPROC. Db2 uses this package to process the SQL statements in this procedure.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
  ...
 SET CURRENT PACKAGESET = 'COLL2'
  ...
END

BIND PACKAGE(COLL2) COPY(TEST.MYPROC) COPYVER(ABC)
                    ACTION(ADD) QUALIFIER(XYZ)
```

Related tasks

Regenerating an existing version of a native SQL procedure
When you apply Db2 maintenance that changes how native SQL procedures are generated, you need to regenerate any affected procedures. When you regenerate a version of a native SQL procedure, Db2 rebinds the associated package for that version of the procedure.

Replacing copies of a package for a version of a native SQL procedure
When you change a version of a native SQL procedure and the ALTER PROCEDURE REPLACE statement contains certain options, you must replace any local or remote copies of the package that exist for that version of the procedure.

Related reference

ALTER PROCEDURE (SQL - native) (Db2 SQL)

Replacing copies of a package for a version of a native SQL procedure
When you change a version of a native SQL procedure and the ALTER PROCEDURE REPLACE statement contains certain options, you must replace any local or remote copies of the package that exist for that version of the procedure.

About this task

If you specify any of the following ALTER PROCEDURE options, you must replace copies of the package:

- REPLACE VERSION
- REGENERATE
- DISABLE DEBUG MODE
- QUALIFIER
- PACKAGE OWNER
- DEFER PREPARE
- NODEFER PREPARE
- CURRENT DATA
- DEGREE
- DYNAMICRULES
- APPLICATION ENCODING SCHEME
- WITH EXPLAIN
- WITHOUT EXPLAIN
- WITH IMMEDIATE WRITE

- WITHOUT IMMEDIATE WRITE
- ISOLATION LEVEL
- WITH KEEP DYNAMIC
- WITHOUT KEEP DYNAMIC
- OPTHINT
- SQL PATH
- RELEASE AT COMMIT
- RELEASE AT DEALLOCATE
- REOPT
- VALIDATE RUN
- VALIDATE BIND
- ROUNDING
- DATE FORMAT
- DECIMAL
- FOR UPDATE CLAUSE OPTIONAL
- FOR UPDATE CLAUSE REQUIRED
- TIME FORMAT

To replace copies of a package for a version of a native SQL procedure, specify the BIND COPY ACTION(REPLACE) command with the appropriate package name and version ID.

Creating new versions of native SQL procedures

A new version of a native SQL procedure can have different parameter names, procedure options, or procedure body.

About this task

All versions of a procedure must have the same procedure signature. Therefore, each version of the procedure must have the same of the following items:

- Schema name
- Procedure name
- Number of parameters
- Data types for corresponding parameters

When any single version of a procedure is defined as autonomous, all versions must be defined as autonomous.

Important: Do not create additional versions of procedures that are supplied with Db2 by specifying the VERSION keyword. Only versions that are supplied with Db2 are supported. Additional versions of such routines cause the installation and configuration of the supplied routines to fail.

Procedure

To create a new version of a procedure, issue the following:

- The ALTER PROCEDURE statement with the following items:
  - The name of the native SQL procedure for which you want to create a new version.
  - The ADD VERSION clause with a name for the new version.
  - The parameter list of the procedure that you want to alter. This parameter list must be the same as the original procedure.

– Any procedure options. These options can be different than the options for other versions of this procedure. If you do not specify a value for a particular option, the default value is used, regardless of the value that is used by the current active version of this procedure.
– A procedure body. This body can be different than the procedure body for other versions of this procedure.

Examples

Example 1

For example, the following CREATE PROCEDURE statement defines a new native SQL procedure called UPDATE_BALANCE. The version of the procedure is V1, and it is the active version.

```
CREATE PROCEDURE
UPDATE_BALANCE
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
VERSION V1
LANGUAGE SQL
READS SQL DATA
BEGIN
DECLARE CUSTOMER_NAME CHAR(20);
SELECT CUSTNAME
INTO CUSTOMER_NAME
FROM ACCOUNTS
WHERE CUSTNO = CUSTOMER_NO;
END
```

Example 2

The following ALTER PROCEDURE statement creates a new version of the UPDATE_BALANCE procedure. The version name of the new version is V2. This new version has a different procedure body.

```
ALTER PROCEDURE
UPDATE_BALANCE
ADD VERSION V2
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL (9,2) )
MODIFIES SQL DATA
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
WHERE CUSTNO = CUSTOMER_NO;
END
```

What to do next

After you create a new version, if you want that version to be invoked by all subsequent calls to this procedure, you need to make that version the active version.

Related reference

ALTER PROCEDURE (SQL - native) (Db2 SQL)
CREATE PROCEDURE (SQL - native) (Db2 SQL)

Multiple versions of native SQL procedures

You can define multiple versions of a native SQL procedure. Db2 maintains this version information for you.

One or more versions of a procedure can exist at any point in time at the current server, but only one version of a procedure is considered the active version. When you first create a procedure, that initial version is considered the active version of the procedure.

Using multiple versions of a native SQL procedure has the following advantages:

- You can keep the existing version of a procedure active while you create another version. When the other version is ready, you can make it the active one.

- When you make another version of a procedure active, you do not need to change any existing calls to that procedure.
- You can easily switch back to a previous version of a procedure if the version that you switched to does not work as planned.
- You can drop an unneeded version of a procedure.

A new version of a native SQL procedure can have different values for the following items:

- Parameter names
- Procedure options (except for the AUTONOMOUS option, which must be specified for all versions or none)
- Procedure body

Restrictions:

- A new version of a native SQL procedure cannot have different values for the following items:
  - Number of parameters
  - Parameter data types
  - Parameter attributes for character data
  - Parameter CCSIDs
  - Whether a parameter is an input or output parameter, as defined by the IN, OUT, and INOUT options

  If you need to specify different values for any of the preceding items, create a new native SQL procedure, instead of a new version.
- When the AUTONOMOUS option is specified for one version of a procedure, it must be specified for every version of that procedure.

Deploying a native SQL procedure to another Db2 for z/OS server

When deploying a native SQL procedure to another Db2 for z/OS server, you can change the bind options to better match the deploying environment. The procedure logic remains the same.

Before you begin

Deprecated function: The DEPLOY bind option is deprecated. For best results, deploy compiled SQL functions and native SQL procedures to multiple environments by issuing the same CREATE or ALTER statements separately in each Db2 environment.

Requirements:

- The remote server must be properly defined in the communications database of the Db2 subsystem from which you deploy the native SQL procedure.
- The target Db2 subsystem must be operating at a PTF level that is compatible with the PTF level of the local Db2 subsystem.

Procedure

Issue the BIND PACKAGE command with the following options:

DEPLOY
Specify the name of the procedure whose logic you want to use on the target server.

Tip: When specifying the parameters for the DEPLOY option, consider the following naming rules for native SQL procedures:

- The collection ID is the same as the schema name in the original CREATE PROCEDURE statement.
- The package ID is the same as the procedure name in the original CREATE PROCEDURE statement.

COPYVER
Specify the version of the procedure whose logic you want to use on the target server.

ACTION(ADD) or ACTION(REPLACE)
: Specify whether you want Db2 to create a new version of the native SQL procedure and its associated package or to replace the specified version.

Optionally, you can also specify the bind options QUALIFIER or OWNER if want to change them.

Examples

Deploying the same version of a procedure at another location
: The following BIND command creates a native SQL procedure with the name PRODUCTION.MYPROC at the CHICAGO location. This procedure is created from the procedure TEST.MYPROC at the current site. Both native SQL procedures have the same content and version, ABC. However, the package for the procedure CHICAGO.PRODUCTION.MYPROC has XYZ as its qualifier.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
 ...
END

BIND PACKAGE(CHICAGO.PRODUCTION) DEPLOY(TEST.MYPROC) COPYVER(ABC)
                                 ACTION(ADD) QUALIFIER(XYZ)
```

Replacing a version of a procedure at another location
: The following BIND command replaces version ABC of the procedure PRODUCTION.MYPROC at the CHICAGO location with version ABC of the procedure TEST.MYPROC at the current site.

```
BIND PACKAGE(CHICAGO.PRODUCTION) DEPLOY(TEST.MYPROC) COPYVER(ABC)
                                 ACTION(REPLACE) REPLVER(ABC)
```

Related concepts

Communications database for the server (Managing Security)

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

BIND PACKAGE (DSN) (Db2 Commands)

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Removing an existing version of a native SQL procedure

You can drop a particular version of a native SQL procedure without dropping the other versions of the procedure.

Before you begin

Before you remove an existing version of a native SQL procedure, ensure that the version is not active. If the version is the active version, designate a different active version before proceeding.

Procedure

Issue the ALTER PROCEDURE statement with the DROP VERSION clause and the name of the version that you want to drop. If you instead want to drop all versions of the procedure, use the DROP statement.

Examples

Example of dropping a version that is not active
: The following statement drops the OLD_PRODUCTION version of the P1 procedure.

```
ALTER PROCEDURE P1 DROP VERSION OLD_PRODUCTION
```

Example of dropping an active version
: Assume that the OLD_PRODUCTION version of the P1 procedure is the active version. The following example first switches the active version to NEW_PRODUCTION and then drops the OLD_PRODUCTION version.

```
ALTER PROCEDURE P1 ACTIVATE VERSION NEW_PRODUCTION;
ALTER PROCEDURE P1 DROP VERSION OLD_PRODUCTION;
```

Related tasks

Designating the active version of a native SQL procedure
When a native SQL procedure is called, Db2 uses the version that is designated as the active version.

Regenerating an existing version of a native SQL procedure

When you apply Db2 maintenance that changes how native SQL procedures are generated, you need to regenerate any affected procedures. When you regenerate a version of a native SQL procedure, Db2 rebinds the associated package for that version of the procedure.

About this task

ALTER PROCEDURE REGENERATE is different than the REBIND PACKAGE command. When you specify REBIND PACKAGE, Db2 rebinds only the non-control SQL statements. Use this command when you think rebinding will improve the access path. When you specify ALTER PROCEDURE REGENERATE, Db2 rebinds the SQL control statements as well as the non-control statements.

Procedure

To regenerate an existing version of a native SQL procedure:

1. Issue the ALTER PROCEDURE statement with the REGENERATE clause and specify the version to be regenerated.
2. If copies of the package for the specified version of the procedure exist at remote sites, replace those packages. Issue the BIND PACKAGE command with the COPY option and appropriate location for each remote package.
3. If copies of the package for the specified version of the procedure exist locally with different schema names, replace those packages. Issue the BIND PACKAGE command with the COPY option and appropriate schema for each local package.

Example

The following ALTER PROCEDURE statement regenerates the active version of the UPDATE_SALARY_1 procedure.

```
ALTER PROCEDURE UPDATE_SALARY_1
REGENERATE ACTIVE VERSION
```

Changing an existing version of a native SQL procedure

You can change an option or the procedure body for a particular version of a native SQL procedure. If you want to keep a copy of that stored procedure, consider creating a new version instead of changing the existing version.

Procedure

To change an existing version of a native SQL procedure, issue the following statement:

- The ALTER PROCEDURE statement with the REPLACE VERSION clause.

  Any option that you do not explicitly specify inherits the system default values. This inheritance occurs even if those options were explicitly specified for a prior version by using a CREATE PROCEDURE statement, ALTER PROCEDURE statement, or REBIND command.

Examples

Example 1

The following ALTER PROCEDURE statement updates version V2 of the UPDATE_BALANCE procedure.

```
ALTER PROCEDURE
TEST.UPDATE_BALANCE
REPLACE VERSION V2
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
MODIFIES SQL DATA
ASUTIME LIMIT 100
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
WHERE CUSTNO = CUSTOMER_NO
AND CUSTSTAT = 'A';
END
```

Related tasks

Creating new versions of native SQL procedures
A new version of a native SQL procedure can have different parameter names, procedure options, or procedure body.

Related reference

REBIND PACKAGE (DSN) (Db2 Commands)

ALTER PROCEDURE (SQL - native) (Db2 SQL)

CREATE PROCEDURE (SQL - native) (Db2 SQL)

Creating external stored procedures

An *external stored procedure* is a procedure that is written in a host language and can contain SQL statements. The source code for external procedures is separate from the definition.

Before you begin

Before you create an external procedure, Configure Db2 for running stored procedures and user-defined functions during installation or Configure Db2 for running stored procedures and user-defined functions during migration.

About this task

Restriction: These instructions do not apply to Java stored procedures. The process for creating a Java stored procedure is different. The preparation process varies depending on what the procedure contains.

Procedure

To create an external stored procedure:

1. Write the external stored procedure body in assembler, C, C++, COBOL, REXX, or PL/I.

   Ensure that the procedure body that you write follows the guidelines for external stored procedures that are described in the following information:

   - “Accessing other sites in an external procedure” on page 276
   - “Accessing non-Db2 resources in your stored procedure” on page 277
   - “Writing an external procedure to access IMS databases” on page 278
   - “Writing an external procedure to return result sets to a distributed client” on page 279
   - “Restrictions when calling other programs from an external stored procedure” on page 280
   - “External stored procedures as main programs and subprograms” on page 281
   - “Data types in stored procedures” on page 283
   - “COMMIT and ROLLBACK statements in a stored procedure” on page 230

   Restrictions:

   - Do not include explicit attachment facility calls. External stored procedures that run in a WLM-established address space use Resource Recovery Services attachment facility (RRSAF) calls implicitly. If an external stored procedure makes an explicit attachment facility call, Db2 rejects the call.

- Do not include SRRCMIT or SRRBACK service calls. If an external stored procedure invokes either SRRCMIT or SRRBACK, Db2 puts the transaction in a state where a rollback operation is required and the CALL statement fails.

For REXX procedures, continue with step "3" on page 257.

2. For assembler, C, C++, COBOL, or PL/I stored procedures, prepare the external procedure by completing the following tasks:
   a) Precompile, compile, and link-edit the application by using one of the following techniques:
      - The Db2 precompiler and JCL instructions to compile and link-edit the program
      - The SQL statement coprocessor

      Recommendation: Compile and link-edit code as reentrant.

      Link-edit the application by using DSNRLI, the language interface module for the Resource Recovery Services attachment facility, or DSNULI, the Universal language interface module. You must specify the parameter AMODE(31) when you link-edit the application with either of these modules. (24-bit applications are not supported.)

      If you want to make the stored procedure reentrant, see "Creating an external stored procedure as reentrant" on page 280

      If you want to run your procedure as a z/OS-authorized program, you must also perform the following tasks when you link-edit the application:
      - Indicate that the load module can use restricted system services by specifying the parameter value AC=1.
      - Put the load module for the stored procedure in an APF-authorized library.

      You can compile COBOL stored procedures with either the DYNAM or NODYNAM COBOL compiler options. If you use DYNAM, ensure that the correct Db2 language interface module is loaded dynamically by performing one of the following actions:
      - Specify the ATTACH(RRSAF) SQL processing option.
      - Copy the DSNRLI module into a load library that is concatenated in front of the Db2 libraries. Use the member name DSNHLI.
   b) Bind the DBRM into a Db2 package by issuing the BIND PACKAGE command.

      If you want to control access to a stored procedure package, specify the ENABLE bind option with the system connection type of the calling application.

      Stored procedures require only a package. You do not need to bind a plan for the stored procedure or bind the stored procedure package to the plan for the calling application. For remote access scenarios, you need a package at both the requester and server sites.

      For more information about stored procedure packages, see "Packages for external stored procedures" on page 275.

      The following example BIND PACKAGE command binds the DBRM EMPDTL1P to the collection DEVL7083.

```
BIND PACKAGE(DEVL7083) -
MEMBER(EMPDTL1P) ACT(REP) ISO(UR) ENCODING(EBCDIC) -
OWNER(DEVL7083) LIBRARY('SG247083.DEVL.DBRM')
```

3. Define the stored procedure to Db2 by issuing the CREATE PROCEDURE statement with the EXTERNAL option. Use the EXTERNAL NAME clause to specify the name of the load module for the program that runs when this procedure is called.

If you want to run your procedure as a z/OS-authorized program, specify an appropriate environment with the WLM ENVIRONMENT option. The stored procedure must run in an address space with a startup procedure in which all libraries in the STEPLIB concatenation are APF-authorized.

If you want environment information to be passed to the stored procedure when it is invoked, specify the DBINFO and PARAMETER STYLE SQL options in the CREATE PROCEDURE statement. When the procedure is invoked, Db2 passes the DBINFO structure, which contains environment information, to the stored procedure. For more information about PARAMETER STYLE, see "Defining the linkage convention for an external stored procedure" on page 259.

If you compiled the stored procedure as reentrant, specify the STAY RESIDENT YES option in the CREATE PROCEDURE statement. This option makes the procedure remain resident in storage.

4. Authorize the appropriate users to use the stored procedure by issuing the GRANT EXECUTE statement.
For example, the following statement allows an application that runs under the authorization ID JONES to call stored procedure SPSCHEMA.STORPRCA:

```
GRANT EXECUTE ON PROCEDURE SPSCHEMA.STORPRCA TO JONES;
```

Example of defining a C stored procedure

Suppose that you have written and prepared a stored procedure that has the following characteristics:

- The name of the stored procedure is B.
- The stored procedure has the following two parameters:
  - An integer input parameter that is named V1
  - A character output parameter of length 9 that is named V2
- The stored procedure is written in the C language.
- The stored procedure contains no SQL statements.
- The same input always produces the same output.
- The load module name is SUMMOD.
- The package collection name is SUMCOLL.
- The stored procedure is to run for no more than 900 CPU service units.
- The parameters can have null values.
- The stored procedure is to be deleted from memory when it completes.
- The stored procedure needs the following Language Environment runtime options:

```
MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)
```

- The stored procedure is part of the WLM application environment that is named PAYROLL.
- The stored procedure runs as a main program.
- The stored procedure does not access non-Db2 resources, so it does not need a special RACF environment.
- The stored procedure can return at most 10 result sets.
- When control returns to the client program, Db2 does not commit updates automatically.

The following CREATE PROCEDURE statement defines the stored procedure to Db2:

```
CREATE PROCEDURE B(IN V1 INTEGER, OUT V2 CHAR(9))
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME SUMMOD
  COLLID SUMCOLL
  ASUTIME LIMIT 900
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
```

```
   RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
   WLM ENVIRONMENT PAYROLL
   PROGRAM TYPE MAIN
   SECURITY DB2
   DYNAMIC RESULT SETS 10
   COMMIT ON RETURN NO;
```

What to do next

You can now invoke the stored procedure from an application program or command line processor.

Related concepts

"Universal language interface (DSNULI)" on page 117

The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

Java stored procedures and user-defined functions (Db2 Application Programming for Java)

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

CREATE PROCEDURE (external) (Db2 SQL)

GRANT (function or procedure privileges) (Db2 SQL)

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Defining the linkage convention for an external stored procedure

A linkage convention specifies the rules for the parameter list that is passed by the program that calls the external stored procedure. For example, the convention can specify whether the calling program can pass null values for input parameters.

Procedure

When you define the stored procedure with the CREATE PROCEDURE statement, specify one of the following values for the PARAMETER STYLE option:

- GENERAL
- GENERAL WITH NULLS
- SQL

SQL is the default.

Linkage conventions for external stored procedures

The linkage convention for a stored procedure can be either GENERAL, GENERAL WITH NULLS, or SQL. These linkage conventions apply to only external stored procedures.

GENERAL

Specify the GENERAL linkage convention when you do not want the calling program to pass null values for input parameters (IN or INOUT) to the stored procedure. If you specify GENERAL, ensure that the stored procedure contains a variable declaration for each parameter that is passed in the CALL statement.

The following figure shows the structure of the parameter list for PARAMETER STYLE GENERAL.

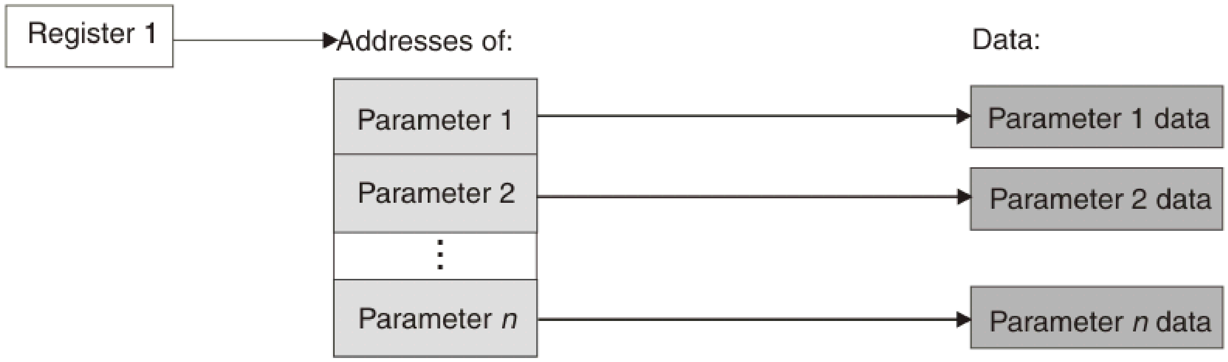


Figure 12. Parameter convention GENERAL for a stored procedure

GENERAL WITH NULLS

Specify the GENERAL WITH NULLS linkage convention when you want to allow the calling program to supply a null value for any parameter that is passed to the stored procedure. If you specify GENERAL WITH NULLS, ensure that the stored procedure performs the following tasks:

- Declares a variable for each parameter that is passed in the CALL statement.
- Declares a null indicator structure that contains an indicator variable for each parameter.
- On entry, examines all indicator variables that are associated with input parameters to determine which parameters contain null values.
- On exit, assigns values to all indicator variables that are associated with output variables. If the output variable returns a null value to the caller, assign the associated indicator variable a negative number. Otherwise, assign a value of 0 to the indicator variable.

In the CALL statement in the calling application, follow each parameter with its indicator variable. Use one of the following forms:

- *host-variable* :*indicator-variable*
- *host-variable* INDICATOR :*indicator-variable*

The following figure shows the structure of the parameter list for PARAMETER STYLE GENERAL WITH NULLS.

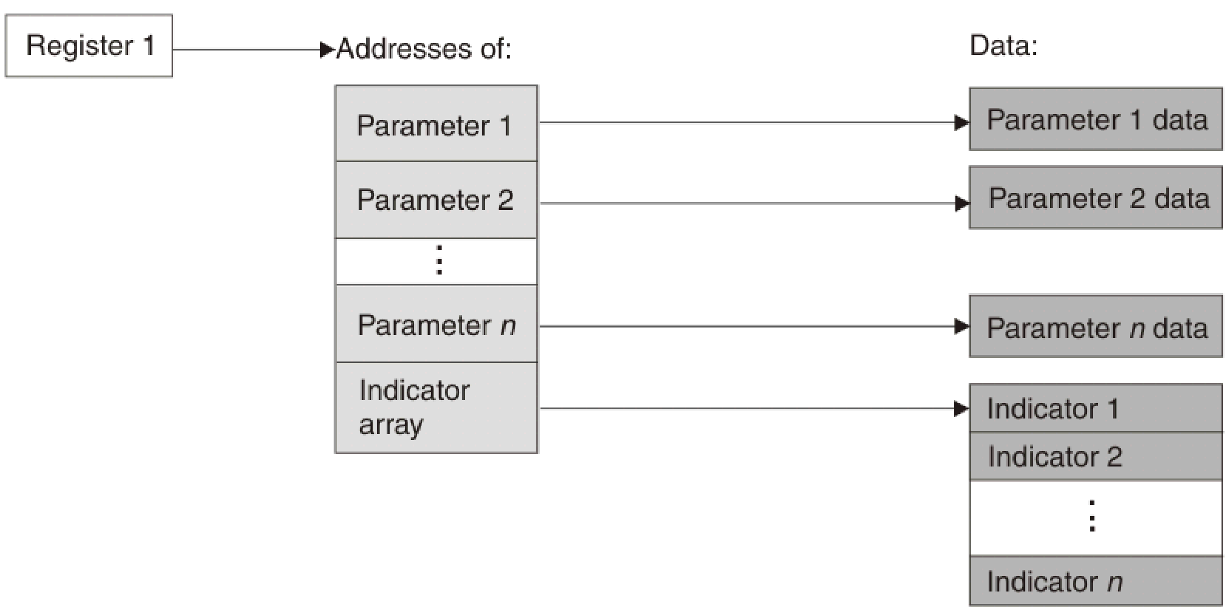


Figure 13. Parameter convention GENERAL WITH NULLS for a stored procedure

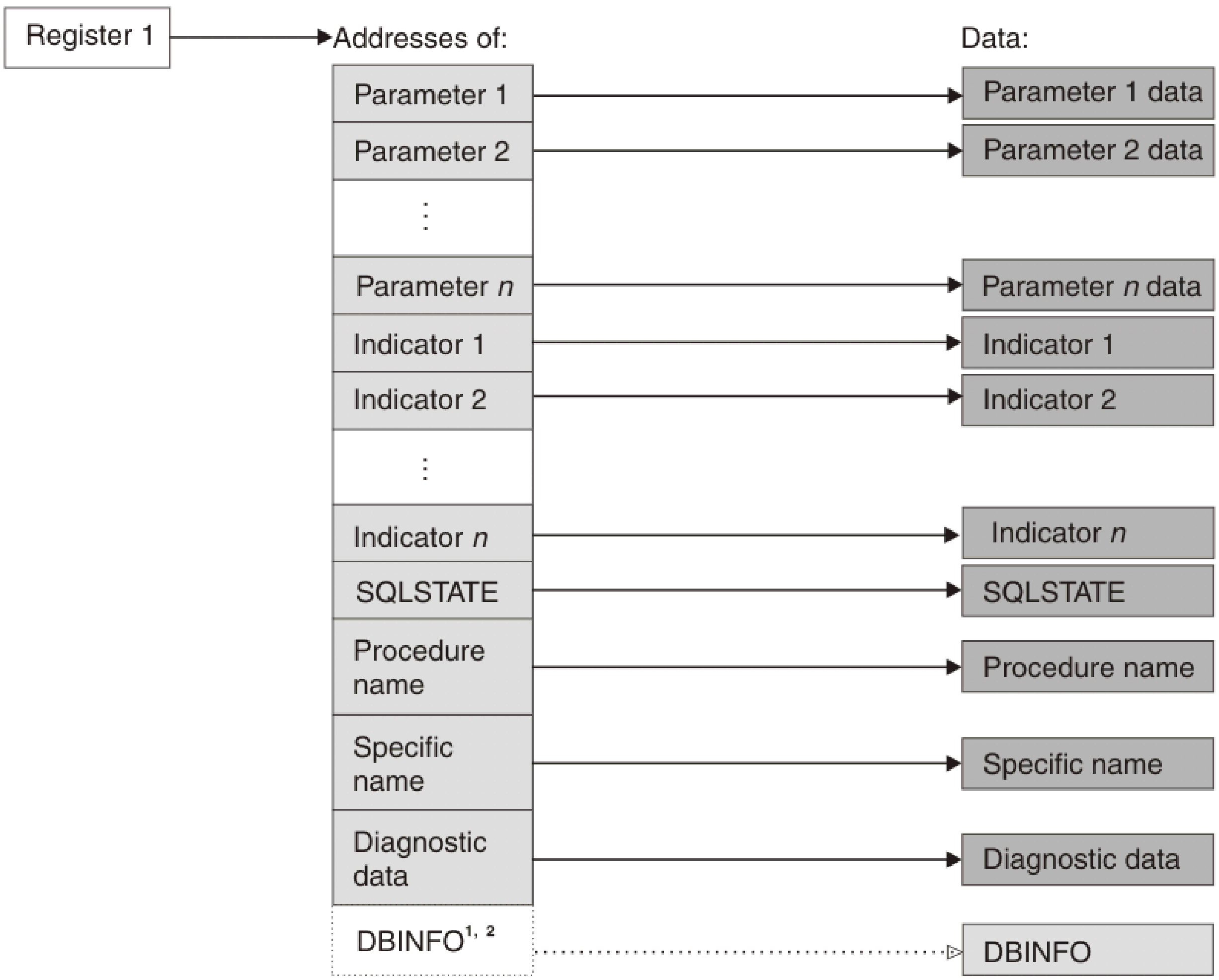
SQL

Specify the SQL linkage convention when you want both of the following conditions:

- The calling program to be able to supply a null value for any parameter that is passed to the stored procedure.
- Db2 to pass input and output parameters to the stored procedure that contain the following information:
  - The SQLSTATE that is to be returned to Db2. This value is a CHAR(5) parameter that represents the SQLSTATE that is passed into the program from the database manager. The initial value is set to '00000'. Although the SQLSTATE is usually not set by the program, it can be set as the result SQLSTATE that is used to return an error or a warning. Returned values that start with anything other than '00', '01', or '02' are error conditions.
  - The qualified name of the stored procedure. This is a VARCHAR(128) value.
  - The specific name of the stored procedure. The specific name is a VARCHAR(128) value that is the same as the unqualified name.
  - The SQL diagnostic string that is to be returned to Db2. This is a VARCHAR(1000) value. Use this area to pass descriptive information about an error or warning to the caller.

Restriction: You cannot use the SQL linkage convention for a REXX language stored procedure.

The following figure shows the structure of the parameter list for PARAMETER STYLE SQL.



[1] For PL/I, this value is the address of a pointer to the DBINFO data.
[2] Passed if the DBINFO option is specified in the user-defined function definition

Figure 14. Parameter convention SQL for a stored procedure

Related concepts

Example programs that call stored procedures
Examples can be used as models when you write applications that call stored procedures. In addition, *prefix*.SDSNSAMP contains sample jobs DSNTEJ6P and DSNTEJ6S and programs DSN8EP1 and DSN8EP2, which you can run.

Related reference

CREATE PROCEDURE (external) (Db2 SQL)

SQLSTATE values and common error codes (Db2 Codes)

Example of GENERAL linkage convention
Specify the GENERAL linkage convention when you do not want the calling program to pass null values for input parameters (IN or INOUT) to the stored procedure.

Examples

The following examples demonstrate how an assembler, C, COBOL, or PL/I stored procedure uses the GENERAL linkage convention to receive parameters.

For these examples, assume that a COBOL application has the following parameter declarations and CALL statement:

```
************************************************************
* PARAMETERS FOR THE SQL STATEMENT CALL                    *
************************************************************
01 V1  PIC S9(9) USAGE COMP.
01 V2  PIC X(9).
⋮
      EXEC SQL CALL A (:V1, :V2) END-EXEC.
```

In the CREATE PROCEDURE statement, the parameters are defined as follows:

```
IN V1 INT, OUT V2 CHAR(9)
```

Assembler example

The following example shows how a stored procedure that is written in assembler language receives these parameters.

```
*******************************************************************
*  CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES      *
*  THE GENERAL LINKAGE CONVENTION.                                 *
*******************************************************************
A        CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
         USING PROGAREA,R13
*******************************************************************
*  BRING UP THE LANGUAGE ENVIRONMENT.                             *
*******************************************************************
         ⋮
*******************************************************************
*  GET THE PASSED PARAMETER VALUES. THE GENERAL LINKAGE CONVENTION*
*  FOLLOWS THE STANDARD ASSEMBLER LINKAGE CONVENTION:             *
*  ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS TO THE       *
*  PARAMETERS.                                                    *
*******************************************************************
         L     R7,0(R1)            GET POINTER TO V1
         MVC   LOCV1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF V1
         ⋮
         L     R7,4(R1)            GET POINTER TO V2
         MVC   0(9,R7),LOCV2       MOVE A VALUE INTO OUTPUT VAR V2
         ⋮
         CEETERM  RC=0
*******************************************************************
*  VARIABLE DECLARATIONS AND EQUATES                              *
*******************************************************************
R1       EQU   1                   REGISTER 1
R7       EQU   7                   REGISTER 7
PPA      CEEPPA  ,                 CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,                   PLACE LITERAL POOL HERE
PROGAREA DSECT
```

```
         ORG   *+CEEDSASZ          LEAVE SPACE FOR DSA FIXED PART
LOCV1    DS    F                   LOCAL COPY OF PARAMETER V1
LOCV2    DS    CL9                 LOCAL COPY OF PARAMETER V2
         ⋮
PROGSIZE EQU   *-PROGAREA
         CEEDSA  ,                 MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                 MAPPING OF THE COMMON ANCHOR AREA
         END   A
```

C example

The following figure shows how a stored procedure that is written in the C language receives these parameters.

```
#pragma runopts(PLIST(OS))
#pragma options(RENT)
#include <stdlib.h>
#include <stdio.h>
/*****************************************************************/
/* Code for a C language stored procedure that uses the          */
/* GENERAL linkage convention.                                   */
/*****************************************************************/
main(argc,argv)
  int argc;                         /* Number of parameters passed */
  char *argv[];                     /* Array of strings containing */
                                    /*  the parameter values       */
{
  long int locv1;                   /* Local copy of V1            */
  char locv2[10];                   /* Local copy of V2            */
                                    /*  (null-terminated)          */
  ⋮
  /***************************************************************/
  /* Get the passed parameters. The GENERAL linkage convention   */
  /* follows the standard C language parameter passing           */
  /* conventions:                                                */
  /* - argc contains the number of parameters passed             */
  /* - argv[0]   is a pointer to the stored procedure name       */
  /* - argv[1] to argv[n] are pointers to the n parameters     */
  /*   in the SQL statement CALL.                                */
  /***************************************************************/
  if(argc==3)                       /* Should get 3 parameters:    */
  {                                 /* procname, V1, V2            */
    locv1 = *(int *) argv[1];
                                    /* Get local copy of V1        */
    ⋮
    strcpy(argv[2],locv2);
                                    /* Assign a value to V2        */
  ⋮
  }
}
```

COBOL example

The following figure shows how a stored procedure that is written in the COBOL language receives these parameters.

```
 CBL RENT
 IDENTIFICATION DIVISION.
************************************************************
* CODE FOR A COBOL LANGUAGE STORED PROCEDURE THAT USES THE *
* GENERAL LINKAGE CONVENTION.                              *
************************************************************
 PROGRAM-ID.    A.
 ⋮
 DATA DIVISION.
 ⋮
 LINKAGE SECTION.
************************************************************
* DECLARE THE PARAMETERS PASSED BY THE SQL STATEMENT       *
*  CALL HERE.                                              *
************************************************************
 01 V1  PIC S9(9) USAGE COMP.
 01 V2  PIC X(9).
 ⋮
 PROCEDURE DIVISION USING V1, V2.
************************************************************
```

```
* THE USING PHRASE INDICATES THAT VARIABLES  V1 AND V2     *
*  WERE PASSED BY THE CALLING PROGRAM.                     *
************************************************************
 ⋮
****************************************
* ASSIGN A VALUE TO OUTPUT VARIABLE V2 *
****************************************
   MOVE '123456789' TO V2.
```

PL/I example

The following figure shows how a stored procedure that is written in the PL/I language receives these parameters.

```
*PROCESS SYSTEM(MVS);
A: PROC(V1, V2) OPTIONS(MAIN NOEXECOPS REENTRANT);
 /***************************************************************/
 /* Code for a PL/I language stored procedure that uses the     */
 /* GENERAL linkage convention.                                  */
 /***************************************************************/
 /***************************************************************/
 /* Indicate on the PROCEDURE statement that two parameters     */
 /*  were passed by the SQL statement CALL. Then declare the    */
 /*  parameters in the following section.                       */
 /***************************************************************/
   DCL V1 BIN FIXED(31),
       V2 CHAR(9);
   ⋮
   V2 = '123456789';    /* Assign a value to output variable V2 */
```

Example of GENERAL WITH NULLS linkage convention
Specify the GENERAL WITH NULLS linkage convention when you want to allow the calling program to supply a null value for any parameter that is passed to the stored procedure.

Examples

The following examples demonstrate how an assembler, C, COBOL, or PL/I stored procedure uses the GENERAL WITH NULLS linkage convention to receive parameters.

For these examples, assume that a C application has the following parameter declarations and CALL statement:

```
/************************************************************/
/* Parameters for the SQL statement CALL                    */
/************************************************************/
  long int v1;
  char v2[10];                /* Allow an extra byte for     */
                             /*  the null terminator        */
/************************************************************/
/* Indicator structure                                      */
/************************************************************/
  struct indicators {
    short int ind1;
    short int ind2;
  } indstruc;
  ⋮
  indstruc.ind1 = 0;         /* Remember to initialize the  */
                             /*  input parameter's indicator*/
                             /*  variable before executing  */
                             /*  the CALL statement         */
  EXEC SQL CALL B (:v1 :indstruc.ind1, :v2 :indstruc.ind2);
  ⋮
```

In the CREATE PROCEDURE statement, the parameters are defined as follows:

```
IN V1 INT, OUT V2 CHAR(9)
```

Assembler example

The following figure shows how a stored procedure that is written in assembler language receives these parameters.

```
*******************************************************************
*  CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES      *
*  THE GENERAL WITH NULLS LINKAGE CONVENTION.                     *
*******************************************************************
B        CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
         USING PROGAREA,R13
*******************************************************************
*  BRING UP THE LANGUAGE ENVIRONMENT.                             *
*******************************************************************
         ⋮
*******************************************************************
*  GET THE PASSED PARAMETER VALUES. THE GENERAL WITH NULLS LINKAGE*
*  CONVENTION IS AS FOLLOWS:                                      *
*  ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS. IF N        *
*  PARAMETERS ARE PASSED, THERE ARE N+1 POINTERS. THE FIRST       *
*  N POINTERS ARE THE ADDRESSES OF THE N PARAMETERS, JUST AS      *
*  WITH THE GENERAL LINKAGE CONVENTION. THE N+1ST POINTER IS      *
*  THE ADDRESS OF A LIST CONTAINING THE N INDICATOR VARIABLE      *
*  VALUES.                                                        *
*******************************************************************
         L     R7,0(R1)             GET POINTER TO V1
         MVC   LOCV1(4),0(R7)       MOVE VALUE INTO LOCAL COPY OF V1
         L     R7,8(R1)             GET POINTER TO INDICATOR ARRAY
         MVC   LOCIND(2*2),0(R7)    MOVE VALUES INTO LOCAL STORAGE
         LH    R7,LOCIND            GET INDICATOR VARIABLE FOR V1
         LTR   R7,R7                CHECK IF IT IS NEGATIVE
         BM    NULLIN               IF SO, V1 IS NULL
         ⋮
         L     R7,4(R1)             GET POINTER TO V2
         MVC   0(9,R7),LOCV2        MOVE A VALUE INTO OUTPUT VAR V2
         L     R7,8(R1)             GET POINTER TO INDICATOR ARRAY
         MVC   2(2,R7),=H(0)        MOVE ZERO TO V2'S INDICATOR VAR
         ⋮
         CEETERM  RC=0
*******************************************************************
*  VARIABLE DECLARATIONS AND EQUATES                              *
*******************************************************************
R1       EQU   1                    REGISTER 1
R7       EQU   7                    REGISTER 7
PPA      CEEPPA  ,                  CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,                    PLACE LITERAL POOL HERE
PROGAREA DSECT
         ORG   *+CEEDSASZ           LEAVE SPACE FOR DSA FIXED PART
LOCV1    DS    F                    LOCAL COPY OF PARAMETER V1
LOCV2    DS    CL9                  LOCAL COPY OF PARAMETER V2
LOCIND   DS    2H                   LOCAL COPY OF INDICATOR ARRAY
         ⋮
PROGSIZE EQU   *-PROGAREA
         CEEDSA  ,                  MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                  MAPPING OF THE COMMON ANCHOR AREA
         END   B
```

C example

The following figure shows how a stored procedure that is written in the C language receives these parameters.

```
#pragma options(RENT)
#pragma runopts(PLIST(OS))
#include <stdlib.h>
#include <stdio.h>
/*****************************************************************/
/* Code for a C language stored procedure that uses the          */
/* GENERAL WITH NULLS linkage convention.                        */
/*****************************************************************/
main(argc,argv)
  int argc;                       /* Number of parameters passed */
  char *argv[];                   /* Array of strings containing */
                                  /*  the parameter values       */
{
  long int locv1;                 /* Local copy of V1            */
  char locv2[10];                 /* Local copy of V2            */
                                  /*  (null-terminated)          */
```

```
  short int locind[2];            /* Local copy of indicator     */
                                  /*  variable array             */
  short int *tempint;             /* Used for receiving the      */
                                  /*  indicator variable array   */
  ⋮
  /**************************************************************/
  /* Get the passed parameters. The GENERAL WITH NULLS linkage   */
  /* convention is as follows:                                   */
  /* - argc contains the number of parameters passed             */
  /* - argv[0] is a pointer to the stored procedure name         */
  /* - argv[1] to argv[n] are pointers to the n parameters     */
  /*   in the SQL statement CALL.                                */
  /* - argv[n+1] is a pointer to the indicator variable array    */
  /**************************************************************/
  if(argc==4)                     /* Should get 4 parameters:    */
  {                               /*  procname, V1, V2,          */
                                  /*  indicator variable array   */
    locv1 = *(int *) argv[1];
                                  /* Get local copy of V1        */
    tempint = argv[3];            /* Get pointer to indicator    */
                                  /*  variable array             */
    locind[0] = *tempint;
                                  /* Get 1st indicator variable  */
    locind[1] = *(++tempint);
                                  /* Get 2nd indicator variable  */
    if(locind[0]<0)               /* If 1st indicator variable   */
    {                             /*  is negative, V1 is null    */
      ⋮
    }
    ⋮
    strcpy(argv[2],locv2);
                                  /* Assign a value to V2        */
    *(++tempint) = 0;             /* Assign 0 to V2's indicator  */
                                  /*  variable                   */
  }
}
```

COBOL example

The following figure shows how a stored procedure that is written in the COBOL language receives these parameters.

```
 CBL RENT
 IDENTIFICATION DIVISION.
************************************************************
* CODE FOR A COBOL LANGUAGE STORED PROCEDURE THAT USES THE *
* GENERAL WITH NULLS LINKAGE CONVENTION.                   *
************************************************************
 PROGRAM-ID.    B.
 ⋮
 DATA DIVISION.
 ⋮
 LINKAGE SECTION.
************************************************************
* DECLARE THE PARAMETERS AND THE INDICATOR ARRAY THAT      *
*  WERE PASSED BY THE SQL STATEMENT CALL HERE.             *
************************************************************
 01 V1  PIC S9(9) USAGE COMP.
 01 V2  PIC X(9).
*
 01 INDARRAY.
    10 INDVAR  PIC S9(4) USAGE COMP OCCURS 2 TIMES.
 ⋮
 PROCEDURE DIVISION USING V1, V2, INDARRAY.
************************************************************
* THE USING PHRASE INDICATES THAT VARIABLES  V1, V2, AND   *
*  INDARRAY WERE PASSED BY THE CALLING PROGRAM.            *
************************************************************
 ⋮
***************************
* TEST WHETHER V1 IS NULL *
***************************
  IF INDARRAY(1) < 0
    PERFORM NULL-PROCESSING.
 ⋮
****************************************
* ASSIGN A VALUE TO OUTPUT VARIABLE V2 *
*  AND ITS INDICATOR VARIABLE          *
****************************************
```

```
    MOVE '123456789' TO V2.
    MOVE ZERO TO INDARRAY(2).
```

PL/I example

The following figure shows how a stored procedure that is written in the PL/I language receives these parameters.

```
*PROCESS SYSTEM(MVS);
A: PROC(V1, V2, INDSTRUC) OPTIONS(MAIN NOEXECOPS REENTRANT);
 /***************************************************************/
 /* Code for a PL/I language stored procedure that uses the     */
 /* GENERAL WITH NULLS linkage convention.                      */
 /***************************************************************/
 /***************************************************************/
 /* Indicate on the PROCEDURE statement that two parameters     */
 /*  and an indicator variable structure were passed by the SQL */
 /*  statement CALL. Then declare them in the following section.*/
 /* For PL/I, you must declare an indicator variable structure, */
 /*  not an array.                                              */
 /***************************************************************/
   DCL V1 BIN FIXED(31),
       V2 CHAR(9);
   DCL
       01 INDSTRUC,
          02 IND1 BIN FIXED(15),
          02 IND2 BIN FIXED(15);
   ⋮
   IF IND1 < 0 THEN
    CALL NULLVAL;       /* If indicator variable is negative    */
                        /*  then V1 is null                     */
     ⋮
   V2 = '123456789';    /* Assign a value to output variable V2 */
   IND2 = 0;            /* Assign 0 to V2's indicator variable  */
```

Example of SQL linkage convention
Specify the SQL linkage convention when you want diagnostic information to be passed in the parameters and allow null values.

Examples

The following examples demonstrate how an assembler, C, COBOL, or PL/I stored procedure uses the SQL linkage convention to receive parameters. These examples also show how a stored procedure receives the DBINFO structure.

For these examples, assume that a C application has the following parameter declarations and CALL statement:

```
/************************************************************/
/* Parameters for the SQL statement CALL                    */
/************************************************************/
  long int v1;
  char v2[10];                /* Allow an extra byte for     */
                             /*  the null terminator        */
/************************************************************/
/* Indicator variables                                      */
/************************************************************/
  short int ind1;
  short int ind2;
  ⋮
  ind1 = 0;                  /* Remember to initialize the  */
                             /*  input parameter's indicator*/
                             /*  variable before executing  */
                             /*  the CALL statement         */
  EXEC SQL CALL B (:v1 :ind1, :v2 :ind2);
  ⋮
```

In the CREATE PROCEDURE statement, the parameters are defined as follows:

```
IN V1 INT, OUT V2 CHAR(9)
```

Assembler example

The following figure shows how a stored procedure that is written in assembler language receives these parameters.

```
*******************************************************************
*  CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES
*
*  THE SQL LINKAGE CONVENTION.                                    *
*******************************************************************
B        CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
         USING PROGAREA,R13
*******************************************************************
*  BRING UP THE LANGUAGE ENVIRONMENT.
*
*******************************************************************
         ⋮
*******************************************************************
*  GET THE PASSED PARAMETER VALUES. THE SQL LINKAGE            *
*  CONVENTION IS AS FOLLOWS:
*
*  ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS. IF N
*
*  PARAMETERS ARE PASSED, THERE ARE 2N+4 POINTERS. THE FIRST
*
*  N POINTERS ARE THE ADDRESSES OF THE N PARAMETERS, JUST AS
*
*  WITH THE GENERAL LINKAGE CONVENTION. THE NEXT N POINTERS ARE
*
*  THE ADDRESSES OF THE INDICATOR VARIABLE VALUES.  THE LAST
*
*  4 POINTERS (5, IF DBINFO IS PASSED) ARE THE ADDRESSES OF
*
*  INFORMATION ABOUT THE STORED PROCEDURE ENVIRONMENT AND
*
*  EXECUTION RESULTS.
*
*******************************************************************
         L     R7,0(R1)            GET POINTER TO V1
         MVC   LOCV1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF V1
         L     R7,8(R1)            GET POINTER TO 1ST INDICATOR VARIABLE
         MVC   LOCI1(2),0(R7)      MOVE VALUE INTO LOCAL STORAGE
         L     R7,20(R1)           GET POINTER TO STORED PROCEDURE
NAME
         MVC   LOCSPNM(20),0(R7)   MOVE VALUE INTO LOCAL STORAGE
         L     R7,24(R1)           GET POINTER TO DBINFO
         MVC   LOCDBINF(DBINFLN),0(R7)
*                                  MOVE VALUE INTO LOCAL STORAGE
         LH    R7,LOCI1            GET INDICATOR VARIABLE FOR V1
         LTR   R7,R7               CHECK IF IT IS NEGATIVE
         BM    NULLIN              IF SO, V1 IS NULL
         ⋮
         L     R7,4(R1)            GET POINTER TO V2
         MVC   0(9,R7),LOCV2       MOVE A VALUE INTO OUTPUT VAR V2
         L     R7,12(R1)           GET POINTER TO INDICATOR VAR 2
         MVC   0(2,R7),=H'0'       MOVE ZERO TO V2'S INDICATOR VAR
         L     R7,16(R1)           GET POINTER TO SQLSTATE
         MVC   0(5,R7),=CL5'xxxxx' MOVE xxxxx TO SQLSTATE
         ⋮
         CEETERM  RC=0
```

```
*******************************************************************
*  VARIABLE DECLARATIONS AND EQUATES
*
*******************************************************************
R1       EQU   1                   REGISTER 1
R7       EQU   7                   REGISTER 7
PPA      CEEPPA  ,                 CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,                   PLACE LITERAL POOL HERE
PROGAREA DSECT
         ORG   *+CEEDSASZ          LEAVE SPACE FOR DSA FIXED PART
LOCV1    DS    F                   LOCAL COPY OF PARAMETER V1
LOCV2    DS    CL9                 LOCAL COPY OF PARAMETER V2
LOCI1    DS    H                   LOCAL COPY OF INDICATOR 1
LOCI2    DS    H                   LOCAL COPY OF INDICATOR 2
LOCSQST  DS    CL5                 LOCAL COPY OF SQLSTATE
LOCSPNM  DS    H,CL27              LOCAL COPY OF STORED PROC NAME
LOCSPSNM DS    H,CL18              LOCAL COPY OF SPECIFIC NAME
LOCDIAG  DS    H,CL1000            LOCAL COPY OF DIAGNOSTIC DATA
```

```
LOCDBINF DS    0H                   LOCAL COPY OF DBINFO DATA
DBNAMELN DS    H                    DATABASE NAME LENGTH
DBNAME   DS    CL128                DATABASE NAME
AUTHIDLN DS    H                    APPL AUTH ID LENGTH
AUTHID   DS    CL128                APPL AUTH ID
ASC_SBCS DS    F                    ASCII SBCS CCSID
ASC_DBCS DS    F                    ASCII DBCS CCSID
ASC_MIXD DS    F                    ASCII MIXED CCSID
EBC_SBCS DS    F                    EBCDIC SBCS CCSID
EBC_DBCS DS    F                    EBCDIC DBCS CCSID
EBC_MIXD DS    F                    EBCDIC MIXED CCSID
UNI_SBCS DS    F                    UNICODE SBCS CCSID
UNI_DBCS DS    F                    UNICODE DBCS CCSID
UNI_MIXD DS    F                    UNICODE MIXED CCSID
ENCODE   DS    F                    PROCEDURE ENCODING SCHEME
RESERV0  DS    CL20                  RESERVED
TBQUALLN DS    H                    TABLE QUALIFIER LENGTH
TBQUAL   DS    CL128                TABLE QUALIFIER
TBNAMELN DS    H                    TABLE NAME LENGTH
TBNAME   DS    CL128                TABLE NAME
CLNAMELN DS    H                    COLUMN NAME LENGTH
COLNAME  DS    CL128                COLUMN NAME
RELVER   DS    CL8                  DBMS RELEASE AND VERSION
RESERV1  DS    CL2                  RESERVED
PLATFORM DS    F                    DBMS OPERATING SYSTEM
NUMTFCOL DS    H                    NUMBER OF TABLE FUNCTION COLS USED
RESERV2  DS    CL26                 RESERVED
TFCOLNUM DS    A                    POINTER TO TABLE FUNCTION COL LIST
APPLID   DS    A                    POINTER TO APPLICATION ID
RESERV3  DS    CL20                 RESERVED
DBINFLN  EQU   *-LOCDBINF           LENGTH OF DBINFO
         ⋮
PROGSIZE EQU   *-PROGAREA
         CEEDSA  ,                  MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                  MAPPING OF THE COMMON ANCHOR AREA
         END   B
```

C example

The following figure shows how a stored procedure that is written as a main program in the C language receives these parameters.

```
#pragma runopts(plist(os))
#include <;stdlib.h>
#include <;stdio.h>

main(argc,argv)
  int argc;
  char *argv[];
{
  int parm1;
  short int ind1;
  char p_proc[28];
  char p_spec[19];
  /***************************************************/
  /* Assume that the SQL CALL statement included     */
  /* 3 input/output parameters in the parameter list.*/
  /* The argv vector will contain these entries:     */
  /*      argv[0]         1    contains load module  */
  /*      argv[1-3]       3    input/output parms    */
  /*      argv[4-6]       3    null indicators       */
  /*      argv[7]         1    SQLSTATE variable     */
  /*      argv[8]         1    qualified proc name   */
  /*      argv[9]         1    specific proc name    */
  /*      argv[10]        1    diagnostic string     */
  /*      argv[11]      + 1    dbinfo                */
  /*                   ------                        */
  /*                     12    for the argc variable */
  /***************************************************/
  if argc<>12 {
  ⋮
  /* We end up here when invoked with wrong number of parms */
  }
```

```
  /***************************************************/
  /* Assume the first parameter is an integer.       */
  /* The following code shows how to copy the integer*/
  /* parameter into the application storage.         */
```

```
  /**************************************************/
  parm1 = *(int *) argv[1];
  /**************************************************/
  /* We can access the null indicator for the first  */
  /* parameter on the SQL CALL as follows:           */
  /**************************************************/
  ind1 = *(short int *) argv[4];
  /**************************************************/
  /* We can use the following expression            */
  /* to assign 'xxxxx' to the SQLSTATE returned to   */
  /* caller on the SQL CALL statement.              */
  /**************************************************/
  strcpy(argv[7],"xxxxx/0");
  /**************************************************/
  /* We obtain the value of the qualified procedure  */
  /* name with this expression.                     */
  /**************************************************/
  strcpy(p_proc,argv[8]);
  /**************************************************/
  /* We obtain the value of the specific procedure   */
  /* name with this expression.                     */
  /**************************************************/
  strcpy(p_spec,argv[9]);
  /**************************************************/
  /* We can use the following expression to assign   */
  /* 'yyyyyyyy' to the diagnostic string returned    */
  /* in the SQLDA associated with the CALL statement.*/
  /**************************************************/
  strcpy(argv[10],"yyyyyyyy/0");
⋮
}
```

The following figure shows how a stored procedure that is written as a subprogram in the C language receives these parameters.

```
#pragma linkage(myproc,fetchable)
#include <stdlib.h>
#include <stdio.h>
#include <sqludf.h>

void myproc(*parm1 int,              /* assume INT for PARM1
 */
             parm2 char[11],         /* assume CHAR(10) parm2
*/
⋮
            *p_ind1 short int,       /* null indicator for parm1
 */
            *p_ind2 short int,       /* null indicator for parm2
 */
⋮
            p_sqlstate char[6],      /* SQLSTATE returned to DB2
 */
            p_proc char[28],         /* Qualified stored proc name
*/
            p_spec char[19],         /* Specific stored proc name
 */
            p_diag char[1001],         /* Diagnostic string
   */
            struct sqludf_dbinfo *udf_dbinfo);     /* DBINFO
  */
{
  int l_p1;
  char[11] l_p2;
  short int l_ind1;
  short int l_ind2;
  char[6] l_sqlstate;
  char[28] l_proc;
  char[19] l_spec;
  char[71] l_diag;
  sqludf_dbinfo *ludf_dbinfo;
⋮
  /**************************************************/
  /* Copy each of the parameters in the parameter    */
  /* list into a local variable, just to demonstrate */
  /* how the parameters can be referenced.           */
  /**************************************************/
  l_p1 = *parm1;

  strcpy(l_p2,parm2);
```

```
  l_ind1 = *p_ind1;

  l_ind1 = *p_ind2;

  strcpy(l_sqlstate,p_sqlstate);

  strcpy(l_proc,p_proc);

  strcpy(l_spec,p_spec);

  strcpy(l_diag,p_diag);
  memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));
⋮
}
```

COBOL example

The following figure shows how a stored procedure that is written in the COBOL language receives these parameters.

```
 CBL RENT
 IDENTIFICATION DIVISION.
⋮
 DATA DIVISION.
⋮
 LINKAGE SECTION.
* Declare each of the parameters
 01 PARM1 …
 01 PARM2 …
⋮
* Declare a null indicator for each parameter
 01 P-IND1 PIC S9(4) USAGE COMP.
 01 P-IND2 PIC S9(4) USAGE COMP.
⋮
* Declare the SQLSTATE that can be set by stored proc
 01 P-SQLSTATE PIC X(5).
* Declare the qualified procedure name
 01 P-PROC.
   49 P-PROC-LEN PIC 9(4) USAGE BINARY.
   49 P-PROC-TEXT PIC X(27).
* Declare the specific procedure name
 01 P-SPEC.
   49 P-SPEC-LEN PIC 9(4) USAGE BINARY.
   49 P-SPEC-TEXT PIC X(18).
* Declare SQL diagnostic message token
 01 P-DIAG.
   49 P-DIAG-LEN PIC 9(4) USAGE BINARY.
   49 P-DIAG-TEXT PIC X(1000).
*********************************************************
* Structure used for DBINFO                             *
*********************************************************
01  SQLUDF-DBINFO.
*       Location name length
     05 DBNAMELEN PIC 9(4) USAGE BINARY.
*       Location name
     05 DBNAME PIC X(128).
*       authorization ID length
     05 AUTHIDLEN PIC 9(4) USAGE BINARY.
*       authorization ID
     05 AUTHID PIC X(128).
*       environment CCSID information
     05 CODEPG PIC X(48).
     05 CDPG-DB2 REDEFINES CODEPG.
        10 DB2-CCSIDS OCCURS 3 TIMES.
           15 DB2-SBCS PIC 9(9) USAGE BINARY.
           15 DB2-DBCS PIC 9(9) USAGE BINARY.
           15 DB2-MIXED PIC 9(9) USAGE BINARY.
        10 ENCODING-SCHEME PIC 9(9) USAGE BINARY.
        10 RESERVED PIC X(20).
```

```
* other platform-specific
deprecated CCSID structures not included here
*       schema name length
     05 TBSCHEMALEN PIC 9(4) USAGE BINARY.
*       schema name
     05 TBSCHEMA PIC X(128).
*       table name length
     05 TBNAMELEN PIC 9(4) USAGE BINARY.
```

```
*       table name
     05 TBNAME PIC X(128).
*       column name length
     05 COLNAMELEN PIC 9(4) USAGE BINARY.
*       column name
     05 COLNAME  PIC X(128).
*       product information
     05 VER-REL PIC X(8).
*       reserved
     05 RESD0 PIC X(2).
*       platform type
     05 PLATFORM PIC 9(9) USAGE BINARY.
*       number of entries in the TF column list array (tfcolumn, below)
     05 NUMTFCOL PIC 9(4) USAGE BINARY.
*       reserved
     05 RESD1 PIC X(26).
*       tfcolumn will be allocated dynamically of it is defined
*       otherwise this will be a null pointer
     05 TFCOLUMN USAGE IS POINTER.
*       application identifier
     05 APPL-ID USAGE IS POINTER.
*       reserved
     05 RESD2 PIC X(20).
*
⋮
 PROCEDURE DIVISION USING PARM1, PARM2,
                          P-IND1, P-IND2,
                          P-SQLSTATE, P-PROC, P-SPEC, P-DIAG,
                          SQLUDF-DBINFO.
⋮
```

PL/I example

The following figure shows how a stored procedure that is written in the PL/I language receives these parameters.

```
*PROCESS SYSTEM(MVS);
 MYMAIN: PROC(PARM1, PARM2, ...,
              P_IND1, P_IND2, ...,
              P_SQLSTATE, P_PROC, P_SPEC, P_DIAG, DBINFO)
         OPTIONS(MAIN NOEXECOPS REENTRANT);

 DCL PARM1 ...            /* first parameter  */
 DCL PARM2 ...            /* second parameter */
⋮
 DCL P_IND1 BIN FIXED(15);/* indicator for 1st parm    */
 DCL P_IND2 BIN FIXED(15);/* indicator for 2nd parm    */
⋮
 DCL P_SQLSTATE CHAR(5);  /* SQLSTATE to return to DB2 */
 DCL 01 P_PROC  CHAR(27)  /* Qualified procedure name  */
               VARYING;
 DCL 01 P_SPEC  CHAR(18)  /* Specific stored proc      */
               VARYING;
 DCL 01 P_DIAG  CHAR(1000)  /* Diagnostic string         */
               VARYING;
 DCL DBINFO PTR;
```

```
 DCL 01 SP_DBINFO BASED(DBINFO),
/* Dbinfo            */
       03 UDF_DBINFO_LLEN BIN FIXED(15),    /* location length   */
       03 UDF_DBINFO_LOC  CHAR(128),        /* location name     */
       03 UDF_DBINFO_ALEN BIN FIXED(15),    /* auth ID  length   */
       03 UDF_DBINFO_AUTH CHAR(128),        /* authorization ID  */
       03 UDF_DBINFO_CCSID,          /* CCSIDs for DB2 for z/OS */
        05 R1                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_ASBCS  BIN FIXED(15), /* ASCII SBCS CCSID  */
        05 R2                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_ADBCS  BIN FIXED(15), /* ASCII DBCS CCSID  */
        05 R3                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_AMIXED BIN FIXED(15), /* ASCII MIXED CCSID */
        05 R4                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_ESBCS  BIN FIXED(15), /* EBCDIC SBCS CCSID */
        05 R5                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_EDBCS  BIN FIXED(15), /* EBCDIC DBCS CCSID */
        05 R6                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_EMIXED BIN FIXED(15), /* EBCDIC MIXED CCSID*/
        05 R7                BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_USBCS  BIN FIXED(15), /* Unicode SBCS CCSID
```

```
*/
        05 R8               BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_UDBCS  BIN FIXED(15), /* Unicode DBCS CCSID
*/
        05 R9               BIN FIXED(15), /* Reserved          */
        05 UDF_DBINFO_UMIXED BIN FIXED(15), /* Unicode MIXED CCSID*/
        05 UDF_DBINFO_ENCODE BIN FIXED(31), /* SP encode scheme  */
        05 UDF_DBINFO_RESERV0 CHAR(08),      /* reserved
*/
       03 UDF_DBINFO_SLEN BIN FIXED(15),    /* schema length     */
       03 UDF_DBINFO_SCHEMA CHAR(128),      /* schema name       */
       03 UDF_DBINFO_TLEN BIN FIXED(15),    /* table length      */
       03 UDF_DBINFO_TABLE  CHAR(128),      /* table name        */
       03 UDF_DBINFO_CLEN BIN FIXED(15),    /* column length     */
       03 UDF_DBINFO_COLUMN  CHAR(128),     /* column name       */
       03 UDF_DBINFO_RELVER  CHAR(8),       /* DB2 release level */
       03 UDF_DBINFO_RESERV0 CHAR(2),       /* reserved          */
       03 UDF_DBINFO_PLATFORM BIN FIXED(31), /* database platform*/
       03 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # of TF cols used*/
       03 UDF_DBINFO_RESERV1 CHAR(26),       /* reserved         */
       03 UDF_DBINFO_TFCOLUMN PTR,      /* -> table fun col list */
       03 UDF_DBINFO_APPLID   PTR,      /* -> application id     */
       03 UDF_DBINFO_RESERV2 CHAR(20);       /* reserved         */
⋮
```

DBINFO structure

Use the DBINFO structure to pass environment information to user-defined functions and stored procedures. Some fields in the structure are not used for stored procedures.

DBINFO is a structure that contains information such as the name of the current server, the application run time authorization ID and identification of the version and release of the database manager that invoked the procedure.

The DBINFO structure includes the following information:

Location name length
: An unsigned 2-byte integer field. It contains the length of the location name in the next field.

Location name
: A 128-byte character field. It contains the name of the location to which the invoker is currently connected.

Authorization ID length
: An unsigned 2-byte integer field. It contains the length of the authorization ID in the next field.

Authorization ID
: A 128-byte character field. It contains the authorization ID of the application from which the stored procedure is invoked, padded on the right with blanks. If this stored procedure is nested within other routines (user-defined functions or stored procedures), this value is the authorization ID of the application that invoked the highest-level routine.

Subsystem code page
: A 48-byte structure that consists of 10 integer fields and an eight-byte reserved area. These fields provide information about the CCSIDs of the subsystem from which the stored procedure is invoked.

Table qualifier length
: An unsigned 2-byte integer field. This field contains 0.

Table qualifier
: A 128-byte character field. This field is not used for stored procedures.

Table name length
: An unsigned 2-byte integer field. This field contains 0.

Table name
: A 128-byte character field. This field is not used for stored procedures.

Column name length
: An unsigned 2-byte integer field. This field contains 0.

Column name
: A 128-byte character field. This field is not used for stored procedures.

Product information

An 8-byte character field that identifies the product on which the stored procedure executes.

The format of product identifier values is *pppvvrrm*, where *ppp* is a 3-letter product code (such as DSN for Db2), *vv* is the version, *rr* is the release, and *m* is the modification level. For example, DSN11015 identifies Db2 11 in new-function mode, the value is 'DSN11015'. The product code (*ppp*) is one of the following values:

AQT for IBM Db2 Analytics Accelerator for z/OS
ARI for DB2 Server for VSE & VM
DSN for Db2 for z/OS
HTP for non-secure HTTP URL connections for Db2 native REST services
HTS for secure HTTPS connections for Db2 native REST services
JCC for IBM Data Server Driver for JDBC and SQLJ
QSQ for DB2 for i
SQL for Db2 for Linux, UNIX, and Windows

Modification (*m*) values have the following meanings:

0
Conversion and enabling-new-function modes for migration from DB2 10 (CM10, CM10*, ENFM10, and ENFM10*)

5
New-function mode.

Reserved area
2 bytes.

Operating system
A 4-byte integer field. It identifies the operating system on which the program that invokes the user-defined function runs. The value is one of these:

0
Unknown

1
OS/2

3
Windows

4
AIX

5
Windows NT

6
HP-UX

7
Solaris

8
z/OS

13
Siemens Nixdorf

15
Windows 95

16
SCO UNIX

18
Linux

19
: DYNIX/ptx

24
: Linux for S/390

25
: Linux for IBM Z

26
: Linux/IA64

27
: Linux/PPC

28
: Linux/PPC64

29
: Linux/AMD64

400
: iSeries

Number of entries in table function column list
An unsigned 2-byte integer field. This field contains 0.

Reserved area
26 bytes.

Table function column list pointer
This field is not used for stored procedures.

Unique application identifier
This field is a pointer to a string that uniquely identifies the application's connection to Db2. The string is regenerated at for each connection to Db2.

The string is the LUWID, which consists of a fully-qualified LU network name followed by a period and an LUW instance number. The LU network name consists of a one- to eight-character network ID, a period, and a one- to eight-character network LU name. The LUW instance number consists of 12 hexadecimal characters that uniquely identify the unit of work.

Reserved area
20 bytes.

Packages for external stored procedures

An external stored procedure must have an associated package.

As part of the process of creating an external stored procedure, you prepare the procedure, which means that you precompile, compile, link-edit, and bind the application. The result of this process is a Db2 package. You do not need to create a Db2 plan for an external procedure. The procedure runs under the caller's thread and uses the plan from the client program that calls it.

The calling application can use a Db2 package or plan to execute the CALL statement.

Both the stored procedure package and the calling application plan or package must exist on the server before you run the calling application.

The following figure shows this relationship between a client program and a stored procedure. In the figure, the client program, which was bound into package A, issues a CALL statement to program B. Program B is an external stored procedure in a WLM address space. This external stored procedure was bound into package B.

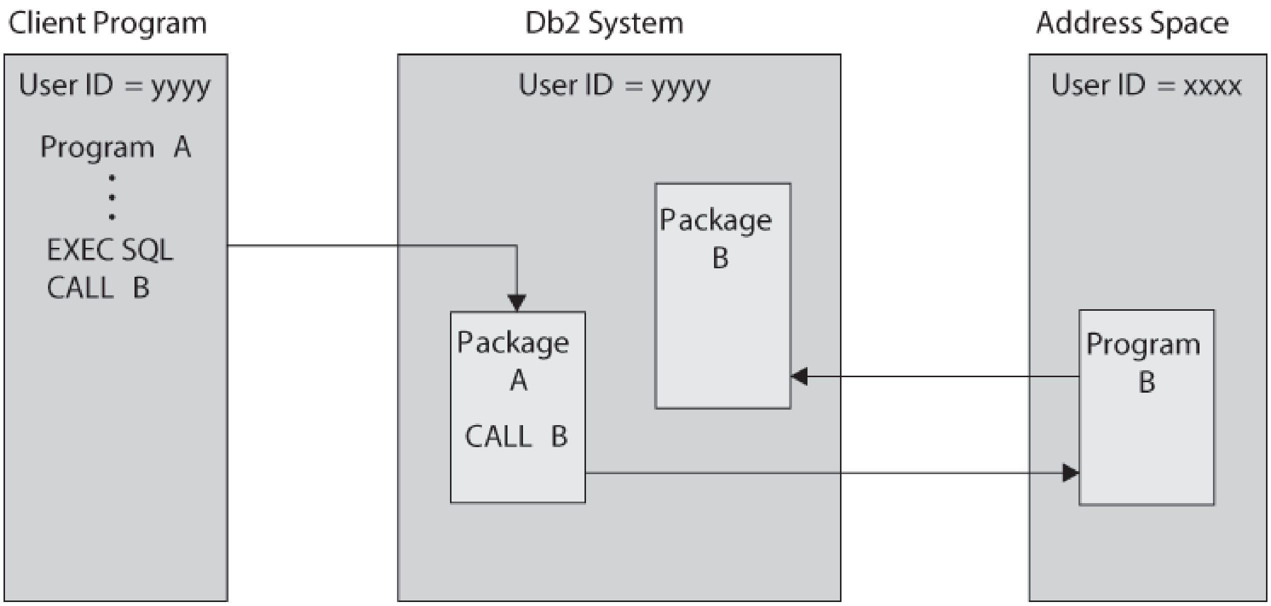


Figure 15. Stored procedure run time environment

You can control access to the stored procedure package by specifying the ENABLE bind option when you bind the package.

In the following situations, the stored procedure might use more than one package:

- You bind a DBRM several times into several versions of the same package, all of which have the same package name but reside in different package collections. Your stored procedure can switch from one version to another by using the SET CURRENT PACKAGESET statement.
- The stored procedure calls another program that contains SQL statements. This program has an associated package. This package must exist at the location where the stored procedure is defined and at the location where the SQL statements are executed.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)
BIND PACKAGE (DSN) (Db2 Commands)
SET CURRENT PACKAGESET (Db2 SQL)

Accessing other sites in an external procedure

External procedures can access tables at other Db2 locations.

About this task

Stored procedures can access tables at other Db2 locations by using three-part object names or CONNECT statements.

Related concepts

Accessing distributed data by using three-part table names

You can use three-part table names to access data at a remote location through DRDA access.

Accessing non-Db2 resources in your stored procedure

Applications that run in a stored procedures address space can access any resources that are available to z/OS address spaces. For example, they can access VSAM files, flat files, APPC/MVS conversations, and IMS or CICS transactions.

About this task

Accessing these resources from a stored procedure can be useful if you want to update older applications. Suppose that you have existing applications that access non-Db2 resources, but you want to use newer Db2 applications to access the same data. You do not need to rewrite the application or migrate the data to Db2. Instead, you can use stored procedures to execute the existing program or access the non-Db2 data directly.

When a stored procedure runs, the stored procedure uses the Recoverable Resource Manager Services (RRS) for commitment control. When Db2 commits or rolls back work, Db2 coordinates all updates that are made to recoverable resources by other RRS compliant resource managers in the z/OS system.

Procedure

To access non-Db2 resources in your stored procedure:

1. Consider serializing access to non-Db2 resources within your application.

   Not all non-Db2 resources can tolerate concurrent access by multiple TCBs in the same address space.
2. To access CICS, use one of the following methods:

   - Stored procedure DSNACICS
   - Message Queue Interface (MQI) for asynchronous execution of CICS transactions
   - External CICS interface (EXCI) for synchronous execution of CICS transactions
   - Advanced Program-to-Program Communication (APPC), using the Common Programming Interface Communications (CPI Communications) application programming interface

   If your system is running a release of CICS that uses z/OS RRS, z/OS RRS controls commitment of all resources.
3. To access IMS DL/I data, use one of the following methods

   - Open Database Access interface (ODBA)
   - Stored procedures DSNAIMS and DSNAIMS2

   If your system is not running a release of IMS that uses z/OS RRS, take one of the following actions:

   - Use the CICS EXCI interface to run a CICS transaction synchronously. That CICS transaction can, in turn, access DL/I data.
   - Invoke IMS transactions asynchronously using the MQI.
   - Use APPC through the Common Programming Interface (CPI) Communications application programming interface.
4. Determine which of the following authorization IDs you want to use to access the non-Db2 resources.

Table 51. Authorization IDs for accessing non-Db2 resources from a stored procedure

ID that you want to use to access the non-Db2 resources	SECURITY value to specify in the CREATE PROCEDURE statement
The authorization ID that is associated with the stored procedures address space	SECURITY Db2
The authorization ID under which the CALL statement is executed	SECURITY USER

Table 51. Authorization IDs for accessing non-Db2 resources from a stored procedure (continued)

ID that you want to use to access the non-Db2 resources	SECURITY value to specify in the CREATE PROCEDURE statement
The authorization ID under which the CREATE PROCEDURE statement is executed	SECURITY DEFINER

5. Issue the CREATE PROCEDURE statement with the appropriate SECURITY option that you determined in the previous step.

Results

When the stored procedure runs, Db2 establishes a RACF environment for accessing non-Db2 resources and uses the specified authorization ID to access protected z/OS resources.

Related tasks

Calling a stored procedure from your application

To run a stored procedure, you can either call it from a client program or invoke it from the command line processor.

Implementing RRS for stored procedures during installation (Db2 Installation and Migration)

Controlling stored procedure access to non-Db2 resources by using RACF (Managing Security)

Related reference

DSNACICS stored procedure (Db2 SQL)

DSNAIMS stored procedure (Db2 SQL)

DSNAIMS2 stored procedure (Db2 SQL)

CREATE PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

APPC/MVS Configuration (Multiplatform APPC Configuration Guide)

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

External CICS interface (EXCI) (CICS Transaction Server for z/OS)

Writing an external procedure to access IMS databases

IMS Open Database Access (ODBA) support lets a Db2 stored procedure connect to an IMS DBCTL or IMS DB/DC system and issue DL/I calls to access IMS databases.

About this task

ODBA support uses RRS for syncpoint control of Db2 and IMS resources. Therefore, stored procedures that use ODBA can run only in WLM-established stored procedures address spaces.

When you write a stored procedure that uses ODBA, follow the rules for writing an IMS application program that issues DL/I calls.

IMS work that is performed in a stored procedure is in the same commit scope as the stored procedure. As with any other stored procedure, the calling application commits work.

A stored procedure that uses ODBA must issue a DPSB PREP call to deallocate a PSB when all IMS work under that PSB is complete. The PREP keyword tells IMS to move inflight work to an indoubt state. When work is in the indoubt state, IMS does not require activation of syncpoint processing when the DPSB call is executed. IMS commits or backs out the work as part of RRS two-phase commit when the stored procedure caller executes COMMIT or ROLLBACK.

A sample COBOL stored procedure and client program demonstrate accessing IMS data using the ODBA interface. The stored procedure source code is in member DSN8EC1 and is prepared by job DSNTEJ61. The calling program source code is in member DSN8EC1 and is prepared and executed by job DSNTEJ62. All code is in data set DSN1110.SDSNSAMP.

The startup procedure for a stored procedures address space in which stored procedures that use ODBA run must include a DFSRESLB DD statement and an extra data set in the STEPLIB concatenation.

Related concepts

Installation step 19: Configure Db2 for running stored procedures and user-defined functions (Db2 Installation and Migration)

Migration step 22: Configure Db2 for running stored procedures and user-defined functions (optional) (Db2 Installation and Migration)

Related information

Application programming design

Writing an external procedure to return result sets to a distributed client

An external procedure can return multiple query result sets to a distributed client if the value of DYNAMIC RESULT SETS in the stored procedure definition is greater than 0.

Procedure

- For each result set you want returned, your stored procedure must complete the following steps:
  a) Declare a cursor with the option WITH RETURN.
  b) Open the cursor.
  c) If the cursor is scrollable, ensure that the cursor is positioned before the first row of the result table.
  d) Leave the cursor open.

  For example, suppose you want to return a result set that contains entries for all employees in department D11. First, declare a cursor that describes this subset of employees:

  ```
  EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
    SELECT * FROM DSN8B10.EMP
      WHERE WORKDEPT='D11';
  ```

  Then, open the cursor:

  ```
  EXEC SQL OPEN C1;
  ```

  Db2 returns the result set and the name of the SQL cursor for the stored procedure to the client.

  When the stored procedure ends, Db2 returns the rows in the query result set to the client.

  Db2 does not return result sets for cursors that are closed before the stored procedure terminates. The stored procedure must execute a CLOSE statement for each cursor associated with a result set that should not be returned to the DRDA client.

- Use meaningful cursor names for returning result sets.

  The name of the cursor that is used to return result sets is made available to the client application through extensions to the DESCRIBE statement.

  Use cursor names that are meaningful to the DRDA client application, especially when the stored procedure returns multiple result sets.

- You can use any of these objects in the SELECT statement that is associated with the cursor for a result set:

  Tables, synonyms, views, created temporary tables, declared temporary tables, and aliases defined at the local Db2 subsystem.

- Return a subset of rows to the client by issuing FETCH statements with a result set cursor. does not return the fetched rows to the client program.

  Db2 does not return the fetched rows to the client program. For example, if you declare a cursor WITH RETURN and then execute the statements OPEN, FETCH, and FETCH, the client receives data beginning with the third row in the result set. If the result set cursor is scrollable and you fetch rows with it, you need to position the cursor before the first row of the result table after you fetch the rows and before the stored procedure ends.

- You can use a created temporary table or declared temporary table to return result sets from a stored procedure.

  This capability can be used to return non-relational data to a DRDA client. For example, you can access IMS data from a stored procedure by using the following process:

  a) Use APPC/MVS to issue an IMS transaction.
  b) Receive the IMS reply message, which contains data that should be returned to the client.
  c) Insert the data from the reply message into a temporary table.
  d) Open a cursor against the temporary table. When the stored procedure ends, the rows from the temporary table are returned to the client.

Related tasks

Writing a program to receive the result sets from a stored procedure
You can write a program to receive results set from a stored procedure for either a fixed number of result sets, for which you know the contents, or a variable number of result sets, for which you do not know the contents.

Restrictions when calling other programs from an external stored procedure

An external procedure can consist of more than one program, each with its own package. Your stored procedure can call other programs, stored procedures, or user-defined functions. Use the facilities of your programming language to call other programs.

If the stored procedure calls other programs that contain SQL statements, each of those called programs must have a Db2 package. The owner of the package or plan that contains the CALL statement must have EXECUTE authority for all packages that the other programs use.

When a stored procedure calls another program, Db2 determines which collection the package of the called program belongs to in one of the following ways:

- If the stored procedure definition contains PACKAGE PATH with a specified list of collection IDs, Db2 uses those collection IDs. If you also specify COLLID, Db2 ignores that clause.
- If the stored procedure definition contains COLLID *collection-id*, Db2 uses *collection-id*.
- If the stored procedure executes SET CURRENT PACKAGE PATH and contains the NO COLLID option, Db2 uses the CURRENT PACKAGE PATH special register. The package of the called program comes from the list of collections in the CURRENT PACKAGE PATH special register. For example, assume that CURRENT PACKAGE PATH contains the list COLL1, COLL2, COLL3, COLL4. Db2 searches for the first package (in the order of the list) that exists in these collections.
- If the stored procedure does not execute SET CURRENT PACKAGE PATH and instead executes SET CURRENT PACKAGESET, Db2 uses the CURRENT PACKAGESET special register. The package of the called program comes from the collection that is specified in the CURRENT PACKAGESET special register.
- If both of the following conditions are true, Db2 uses the collection ID of the package that contains the SQL statement CALL:
  - the stored procedure does not execute SET CURRENT PACKAGE PATH or SET CURRENT PACKAGESET
  - the stored procedure definition contains the NO COLLID option

  When control returns from the stored procedure, the value of the CURRENT PACKAGESET special register is reset.Db2 restores the value of the CURRENT PACKAGESET special register to the value that it contained before the client program executed the SQL statement CALL.

Creating an external stored procedure as reentrant

Reentrant code is code for which a single copy can be used concurrently by two or more processes. For improved performance, prepare your stored procedures to be reentrant whenever possible

About this task

Reentrant stored procedures can improve performance for the following reasons:

- A reentrant stored procedure does not need to be loaded into storage every time that it is called.
- A single copy of the stored procedure can be shared by multiple tasks in the stored procedures address space. This sharing decreases the amount of virtual storage that is used for code in the stored procedures address space.

Procedure

To create an external stored procedure as reentrant:

1. Compile the procedure as reentrant and link-edit it as reentrant and reusable.

   For instructions on compiling programs to be reentrant, see the information for the programming language that you are using. For C and C++ procedures, you can use the z/OS binder to produce reentrant and reusable load modules.

   If your stored procedure cannot be reentrant, link-edit it as non-reentrant and non-reusable. The non-reusable attribute prevents multiple tasks from using a single copy of the stored procedure at the same time.
2. Specify STAY RESIDENT YES in the CREATE PROCEDURE or ALTER PROCEDURE statement for the stored procedure. This option makes a reentrant stored procedure remain in storage.

   A non-reentrant stored procedure must not remain in storage. You therefore need to specify STAY RESIDENT NO in the CREATE PROCEDURE or ALTER PROCEDURE statement for a non-reentrant stored procedure. STAY RESIDENT NO is the default.

Related concepts

Making programs reentrant (Enterprise COBOL for z/OS Programming Guide)

Related reference

Compiler options (COBOL) (Enterprise COBOL for z/OS Programming Guide)
ALTER PROCEDURE (external) (Db2 SQL)
CREATE PROCEDURE (external) (Db2 SQL)
Binder options reference (MVS Program Management: User's Guide and Reference)
Language restricted (Enterprise PL/I for z/OS Compiler and Runtime Migration Guide)
Compile-time option descriptions (PL/I) (Enterprise PL/I for z/OS Programming Guide:)
Reentrancy (XL C/C++ User's Guide)

External stored procedures as main programs and subprograms

A stored procedure that runs in a WLM-established address space and uses Language Environment Release 1.7 or a subsequent release can be either a main program or a subprogram. A stored procedure that runs as a subprogram can perform better because Language Environment does less processing for it.

In general, a subprogram must do the following extra tasks that Language Environment performs for a main program:

- Initialization and cleanup processing
- Allocating and freeing storage
- Closing all open files before exiting

When you code stored procedures as subprograms, follow these rules:

- Follow the language rules for a subprogram. For example, you cannot perform I/O operations in a PL/I subprogram.
- Avoid using statements that terminate the Language Environment enclave when the program ends. Examples of such statements are STOP or EXIT in a PL/I subprogram, or STOP RUN in a COBOL subprogram. If the enclave terminates when a stored procedure ends, and the client program calls another stored procedure that runs as a subprogram, Language Environment must build a new enclave. As a result, the benefits of coding a stored procedure as a subprogram are lost.
- In COBOL stored procedures that are defined as PROGRAM TYPE SUB and STAY RESIDENT YES, if you use stored procedure parameters as host variables, set the SQL-INIT-FLAG variable to 0. This variable

is generated by the Db2 precompiler. Setting it to 0 ensures that the SQLDA is updated with the current addresses.

The following table summarizes the characteristics that define a main program and a subprogram.

Table 52. Characteristics of main programs and subprograms

Language	Main program	Subprogram
Assembler	MAIN=YES is specified in the invocation of the CEEENTRY macro.	MAIN=NO is specified in the invocation of the CEEENTRY macro.
C	Contains a main() function. Pass parameters to it through argc and argv.	A fetchable function. Pass parameters to it explicitly.
COBOL	A COBOL program that ends with GOBACK	A dynamically loaded subprogram that ends with GOBACK
PL/I	Contains a procedure declared with OPTIONS(MAIN)	A procedure declared with OPTIONS(FETCHABLE)

The following code shows an example of coding a C stored procedure as a subprogram.

```
/******************************************************************/
/*  This C subprogram is a stored procedure that uses linkage     */
/*  convention GENERAL and receives 3 parameters.                 */
/******************************************************************/
#pragma linkage(cfunc,fetchable)
#include <stdlib.h>
void cfunc(char p1[11],long *p2,short *p3)
{
  /****************************************************************/
  /*  Declare variables used for SQL operations.  These variables */
  /*  are local to the subprogram and must be copied to and from  */
  /*  the parameter list for the stored procedure call.           */
  /****************************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
  EXEC SQL END DECLARE SECTION;
```

```
  /*************************************************************/
  /*  Receive input parameter values into local variables.     */
  /*************************************************************/
  strcpy(parm1,p1);
  parm2 = *p2;
  parm3 = *p3;
  /*************************************************************/
  /*  Perform operations on local variables.                   */
  /*************************************************************/
  ⋮
  /*************************************************************/
  /*  Set values to be passed back to the caller.              */
  /*************************************************************/
  strcpy(parm1,"SETBYSP");
  parm2 = 100;
  parm3 = 200;
  /*************************************************************/
  /*  Copy values to output parameters.                        */
  /*************************************************************/
  strcpy(p1,parm1);
  *p2 = parm2;
  *p3 = parm3;
}
```

The following code shows an example of coding a C++ stored procedure as a subprogram.

```
/******************************************************************/
/*  This C++ subprogram is a stored procedure that uses linkage   */
/*  convention GENERAL and receives 3 parameters.                 */
/*  The extern statement is required.                             */
/******************************************************************/
extern  "C"  void cppfunc(char p1[11],long *p2,short *p3);
```

```
#pragma linkage(cppfunc,fetchable)
#include <stdlib.h>
EXEC SQL INCLUDE SQLCA;
void cppfunc(char p1[11],long *p2,short *p3)
{
  /****************************************************************/
  /*  Declare variables used for SQL operations.  These variables */
  /*  are local to the subprogram and must be copied to and from  */
  /*  the parameter list for the stored procedure call.           */
  /****************************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
  EXEC SQL END DECLARE SECTION;
```

```
  /*************************************************************/
  /*  Receive input parameter values into local variables.     */
  /*************************************************************/
  strcpy(parm1,p1);
  parm2 = *p2;
  parm3 = *p3;
  /*************************************************************/
  /*  Perform operations on local variables.                   */
  /*************************************************************/
  ⋮
  /*************************************************************/
  /*  Set values to be passed back to the caller.              */
  /*************************************************************/
  strcpy(parm1,"SETBYSP");
  parm2 = 100;
  parm3 = 200;
  /*************************************************************/
  /*  Copy values to output parameters.                        */
  /*************************************************************/
  strcpy(p1,parm1);
  *p2 = parm2;
  *p3 = parm3;
}
```

Data types in stored procedures

A stored procedure that is written in any language except REXX must declare each parameter that is passed to it. The definition for that stored procedure must also contain a compatible SQL data type declaration for each parameter.

For languages other than REXX

For all data types except LOBs, ROWIDs, locators, and VARCHARs (for C language), see the tables listed in the following table for the host data types that are compatible with the data types in the stored procedure definition. You cannot have XML parameters in an external procedure.

For LOBs, ROWIDs, VARCHARs, and locators, the following table shows compatible declarations for the assembler language.

Table 53. Compatible assembler language declarations for LOBs, ROWIDs, and locators

SQL data type in definition	Assembler declaration
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`DS FL4`

Table 53. Compatible assembler language declarations for LOBs, ROWIDs, and locators (continued)

SQL data type in definition	Assembler declaration

BLOB(*n*)	`If n <= 65535:`
	`var DS 0FL4`
	`var_length DS FL4`
	`var_data DS CLn`
	`If n > 65535:`
	`var DS 0FL4`
	`var_length DS FL4`
	`var_data DS CL65535`
	`ORG var_data+(n-65535)`

CLOB(*n*)	`If n <= 65535:`
	`var DS  0FL4`
	`var_length DS FL4`
	`var_data DS CLn`
	`If n > 65535:`
	`var DS 0FL4`
	`var_length DS FL4`
	`var_data DS CL65535`
	`ORG var_data+(n-65535)`

DBCLOB(*n*)	`If m (=2*n) <= 65534:`
	`var DS 0FL4`
	`var_length  DS FL4`
	`var_data DS CLm`
	`If m > 65534:`
	`var DS 0FL4`
	`var_length  DS FL4`
	`var_data DS CL65534`
	`ORG var_data+(m-65534)`

ROWID	`DS HL2,CL40`

VARCHAR(*n*)	If PARAMETER VARCHAR NULTERM is specified or implied:
	`char data[n+1];`
	If PARAMETER VARCHAR STRUCTURE is specified:
	`struct`
	`{short len;`
	` char data[n];`
	`}     var;`

Note:

1. This row does not apply to VARCHAR(*n*) FOR BIT DATA. BIT DATA is always passed in a structured representation.

For LOBs, ROWIDs, and locators, the following table shows compatible declarations for the C language.

Table 54. Compatible C language declarations for LOBs, ROWIDs, and locators

SQL data type in definition	C declaration
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`unsigned long`

SQL data type in definition	C declaration
BLOB(*n*)	`struct {unsigned long length; char data[n]; } var;`
CLOB(*n*)	`struct {unsigned long length; char var_data[n]; } var;`
DBCLOB(*n*)	`struct {unsigned long length; sqldbchar data[n]; } var;`
ROWID	`struct {short int length; char data[40]; } var;`

Table 54. Compatible C language declarations for LOBs, ROWIDs, and locators (continued)

For LOBs, ROWIDs, and locators, the following table shows compatible declarations for COBOL.

Table 55. Compatible COBOL declarations for LOBs, ROWIDs, and locators

SQL data type in definition	COBOL declaration
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`01 var PIC S9(9) COMP-5.`

BLOB(*n*)	`01 var.`
	`49 var-LENGTH PIC S9(9) COMP-5.`
	`49 var-DATA PIC X(n).`

CLOB(*n*)	`01 var.`
	`49 var-LENGTH PIC S9(9) COMP-5.`
	`49 var-DATA PIC X(n).`

DBCLOB(*n*)	`01 var.`
	`49 var-LENGTH PIC S9(9) COMP-5.`
	`49 var-DATA PIC G(n) DISPLAY-1.`

ROWID	`01 var.`
	`49 var-LEN PIC S9(4) COMP-5.`
	`49 var-DATA PIC X(40).`

For LOBs, ROWIDs, and locators, the following table shows compatible declarations for PL/I.

Table 56. Compatible PL/I declarations for LOBs, ROWIDs, and locators

SQL data type in definition	PL/I
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	`BIN FIXED(31)`

BLOB(*n*)	If n <= 32767:
	`01 var,`
	`    03 var_LENGTH`
	`       BIN FIXED(31),`
	`    03 var_DATA`
	`       CHAR(n);`
	If n > 32767:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         CHAR(32767),`
	`      03 var_DATA2`
	`         CHAR(mod(n,32767));`

CLOB(*n*)	If n <= 32767:
	`01 var,`
	`    03 var_LENGTH`
	`       BIN FIXED(31),`
	`    03 var_DATA`
	`       CHAR(n);`
	If n > 32767:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         CHAR(32767),`
	`      03 var_DATA2`
	`         CHAR(mod(n,32767));`

DBCLOB(*n*)	If n <= 16383:
	`01 var,`
	`   03 var_LENGTH`
	`      BIN FIXED(31),`
	`   03 var_DATA`
	`      GRAPHIC(n);`
	If n > 16383:
	`01 var,`
	`   02 var_LENGTH`
	`      BIN FIXED(31),`
	`   02 var_DATA,`
	`      03 var_DATA1(n)`
	`         GRAPHIC(16383),`
	`      03 var_DATA2`
	`         GRAPHIC(mod(n,16383));`

ROWID	`CHAR(40) VAR`

Tables of results: Each high-level language definition for stored procedure parameters supports only a single instance (a scalar value) of the parameter. There is no support for structure, array, or vector parameters. Because of this, the SQL statement CALL limits the ability of an application to return some kinds of tables. For example, an application might need to return a table that represents multiple occurrences of one or more of the parameters passed to the stored procedure. Because the SQL statement CALL cannot return more than one set of parameters, use one of the following techniques to return such a table:

- Put the data that the application returns in a Db2 table. The calling program can receive the data in one of these ways:
  - The calling program can fetch the rows from the table directly. Specify FOR FETCH ONLY or FOR READ ONLY on the SELECT statement that retrieves data from the table. A block fetch can retrieve the required data efficiently.
  - The stored procedure can return the contents of the table as a result set. See "Writing an external procedure to return result sets to a distributed client" on page 279 and "Writing a program to receive the result sets from a stored procedure" on page 768 for more information.
- Convert tabular data to string format and return it as a character string parameter to the calling program. The calling program and the stored procedure can establish a convention for interpreting the content of the character string. For example, the SQL statement CALL can pass a 1920-byte character string parameter to a stored procedure, which enables the stored procedure to return a 24x80 screen image to the calling program.

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

Installation step 19: Configure Db2 for running stored procedures and user-defined functions (Db2 Installation and Migration)

Migration step 22: Configure Db2 for running stored procedures and user-defined functions (optional) (Db2 Installation and Migration)

REXX stored procedures

A REXX stored procedure is similar to any other REXX procedure and follows the same rules as stored procedures in other languages. A REXX stored procedure receives input parameters, executes REXX commands, optionally executes SQL statements, and returns at most one output parameter. However, a few differences exist.

A REXX stored procedure is different from other REXX procedures in the following ways:

- A REXX stored procedure must not execute any of the following DSNREXX commands that are used for the Db2 subsystem thread attachment:

  ```
  ADDRESS DSNREXX CONNECT
  ADDRESS DSNREXX DISCONNECT
  CALL SQLDBS ATTACH TO
  CALL SQLDBS DETACH
  ```

  When you execute SQL statements in your stored procedure, Db2 establishes the connection for you.
- A REXX stored procedure must run in a WLM-established stored procedures address space.
- A language REXX stored procedure executes in a background TSO/E REXX environment provided by the TSO/E environment service IKJTSOEV.

Unlike other stored procedures, you do not prepare REXX stored procedures for execution. REXX stored procedures run using one of four packages that are bound during the installation of Db2 REXX Language Support. The current isolation level at which the stored procedure runs depends on the package that Db2 uses when the stored procedure runs:

Package name
    Isolation level

DSNREXRR
Repeatable read (RR)

DSNREXRS
Read stability (RS)

DSNREXCS
Cursor stability (CS)

DSNREXUR
Uncommitted read (UR)

This topic shows an example of a REXX stored procedure that executes Db2 commands. The stored procedure performs the following actions:

- Receives one input parameter, which contains a Db2 command.
- Calls the IFI COMMAND function to execute the command.
- Extracts the command result messages from the IFI return area and places the messages in a created temporary table. Each row of the temporary table contains a sequence number and the text of one message.
- Opens a cursor to return a result set that contains the command result messages.
- Returns the unformatted contents of the IFI return area in an output parameter.

The following example shows the definition of the stored procedure.

```
CREATE PROCEDURE COMMAND(IN CMDTEXT VARCHAR(254), OUT CMDRESULT VARCHAR(32704))
  LANGUAGE REXX
  EXTERNAL NAME COMMAND
  NO COLLID
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL
  STAY RESIDENT NO
  RUN OPTIONS 'TRAP(ON)'
  WLM ENVIRONMENT WLMENV1
  SECURITY DB2
  DYNAMIC RESULT SETS 1
  COMMIT ON RETURN NO;
```

The following example shows the COMMAND stored procedure that executes Db2 commands.

```
/* REXX */
PARSE UPPER ARG CMD                     /* Get the DB2 command text */
                                        /* Remove enclosing quotation marks  */
IF LEFT(CMD,1) = "'" & RIGHT(CMD,1) = "'" THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
ELSE
IF LEFT(CMD,1) = '"' & RIGHT(CMD,1) = '"' THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
COMMAND    = SUBSTR("COMMAND",1,18," ")
  /****************************************************************/
  /*  Set up the IFCA, return area, and output area for the       */
  /*  IFI COMMAND call.                                           */
  /****************************************************************/
IFCA = SUBSTR('00'X,1,180,'00'X)
IFCA = OVERLAY(D2C(LENGTH(IFCA),2),IFCA,1+0)
IFCA = OVERLAY("IFCA",IFCA,4+1)
RTRNAREASIZE = 262144 /*1048572*/
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'0000'X||CMD
BUFFER = SUBSTR(" ",1,16," ")
  /****************************************************************/
  /*  Make the IFI COMMAND call.                                  */
  /****************************************************************/
ADDRESS LINKPGM "DSNWLIR COMMAND IFCA RTRNAREA OUTPUT"
WRC = RC
RTRN=    SUBSTR(IFCA,12+1,4)
REAS=    SUBSTR(IFCA,16+1,4)
TOTLEN = C2D(SUBSTR(IFCA,20+1,4))
  /****************************************************************/
  /*  Set up the host command environment for SQL calls.          */
  /****************************************************************/
"SUBCOM DSNREXX"                        /* Host cmd env available?  */
```

```
IF RC THEN                             /* No--add host cmd env      */
   S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
```

```
  /****************************************************************/
  /*  Set up SQL statements to insert command output messages     */
  /*  into a temporary table.                                     */
  /****************************************************************/
SQLSTMT='INSERT INTO SYSIBM.SYSPRINT(SEQNO,TEXT) VALUES(?,?)'
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
IF SQLCODE ¬= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQLSTMT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Extract messages from the return area and insert them into  */
  /*  the temporary table.                                        */
  /****************************************************************/
SEQNO = 0
OFFSET = 4+1
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RTRNAREA,OFFSET,2))
  SEQNO = SEQNO + 1
  TEXT = SUBSTR(RTRNAREA,OFFSET+4,LEN-4-1)
  ADDRESS DSNREXX "EXECSQL EXECUTE S1 USING :SEQNO,:TEXT"
  IF SQLCODE ¬= 0 THEN CALL SQLCA
  OFFSET = OFFSET + LEN
END
  /****************************************************************/
  /*  Set up a cursor for a result set that contains the command  */
  /*  output messages from the temporary table.                   */
  /****************************************************************/
SQLSTMT='SELECT SEQNO,TEXT FROM SYSIBM.SYSPRINT ORDER BY SEQNO'
ADDRESS DSNREXX "EXECSQL DECLARE C2 CURSOR FOR S2"
IF SQLCODE ¬= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL PREPARE S2 FROM :SQLSTMT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Open the cursor to return the message output result set to  */
  /*  the caller.                                                 */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL OPEN C2"
IF SQLCODE ¬= 0 THEN CALL SQLCA
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV  */
EXIT SUBSTR(RTRNAREA,1,TOTLEN+4)
```

```
  /****************************************************************/
  /*  Routine to display the SQLCA                                */
  /****************************************************************/
SQLCA:
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
EXIT 'SQLERRMC ='SQLERRMC';' ,
||  'SQLERRP ='SQLERRP';' ,
||  'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6';' ,
||  'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
```

```
             || SQLWARN.2',',
             || SQLWARN.3',',
             || SQLWARN.4',',
             || SQLWARN.5',',
             || SQLWARN.6',',
             || SQLWARN.7',',
             || SQLWARN.8',',
             || SQLWARN.9',',
             || SQLWARN.10';' ,
||  'SQLSTATE='SQLSTATE';'
```

Related reference

Calling a stored procedure from a REXX procedure
The format of the parameters that you pass in the CALL statement in a REXX procedure must be compatible with the data types of the parameters in the CREATE PROCEDURE statement.

TSO/E services available under IKJTSOEV (TSO/E Programming Services)

Modifying an external stored procedure definition

You can modify the definition of an external stored procedure or the stored procedure source code. In either case, you need to prepare the stored procedure again.

Procedure

To modify an external stored procedure definition:

1. Issue the ALTER PROCEDURE statement with the appropriate options.

   This new definition replaces the existing definition.
2. Prepare the external stored procedure again, as you did when you originally created the external stored procedure.

Example

Suppose that an existing C stored procedure was defined with the following statement:

```
CREATE PROCEDURE B(IN V1 INTEGER, OUT V2 CHAR(9))
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME SUMMOD
  COLLID SUMCOLL
  ASUTIME LIMIT 900
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
  RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
  WLM ENVIRONMENT PAYROLL
  PROGRAM TYPE MAIN
  SECURITY DB2
  DYNAMIC RESULT SETS 10
  COMMIT ON RETURN NO;
```

Assume that you need to make the following changes to the stored procedure definition:

- The stored procedure selects data from Db2 tables but does not modify Db2 data.
- The parameters can have null values, and the stored procedure can return a diagnostic string.
- The length of time that the stored procedure runs is unlimited.
- If the stored procedure is called by another stored procedure or a user-defined function, the stored procedure uses the WLM environment of the caller.

The following ALTER PROCEDURE statement makes these changes:

```
ALTER PROCEDURE B
  READS SQL DATA
  ASUTIME NO LIMIT
  PARAMETER STYLE SQL
  WLM ENVIRONMENT (PAYROLL,*);
```

Related tasks

Creating external stored procedures

An *external stored procedure* is a procedure that is written in a host language and can contain SQL statements. The source code for external procedures is separate from the definition.

Related reference

ALTER PROCEDURE (external) (Db2 SQL)

Creating external SQL procedures (deprecated)

An *external SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). However, an external SQL procedure is created, implemented, and executed like other external stored procedures. All SQL procedures that were created prior to DB2 9 are external SQL procedures.

Before you begin

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

Before you create an external SQL procedure, Configure Db2 for running stored procedures and user-defined functions during installation or Configure Db2 for running stored procedures and user-defined functions during migration.

Procedure

To create an external SQL procedure:

1. Use one of the following methods to create the external SQL procedure:
   - IBM Data Studio. See Developing database routines (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio).
   - JCL
   - The Db2 for z/OS SQL procedure processor (DSNTPSMP)

   The preceding methods that you use to create an external SQL procedure perform the following actions:

   - Convert the external SQL procedure source statements into a C language program by using the Db2 precompiler
   - Create an executable load module and a Db2 package from the C language program.
   - Define the external SQL procedure to Db2 by issuing a CREATE PROCEDURE statement either statically or dynamically.

   Restriction: If you plan to use the Db2 stored procedure debugger or the Unified Debugger, do not use JCL. Use either IBM Data Studio or DSNTPSMP.

   If you plan to use IBM Data Studio or DSNTPSMP, you must set up support for external SQL procedures.
2. Authorize the appropriate users to use the stored procedure by issuing the GRANT EXECUTE statement.

Example

For examples of how to prepare and run external SQL procedures, see "Sample programs to help you prepare and run external SQL procedures" on page 307.

Related concepts

SQL procedures

An SQL procedure is a stored procedure that contains only SQL statements.

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Related reference

CREATE PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

GRANT (function or procedure privileges) (Db2 SQL)

Migrating an external SQL procedure to a native SQL procedure

You can migrate an existing external SQL procedure, which is deprecated, to a native SQL procedure by dropping the existing procedure and creating it again as a native SQL procedure. Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

Before you begin

If you created the external SQL procedure in a previous release of Db2, consider the release incompatibilities for applications that use stored procedures, examine your external SQL procedure source code, and make any necessary adjustments. See Application and SQL release incompatibilities (Db2 for z/OS What's New?).

About this task

A *native SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). A native SQL procedure is created by issuing a single SQL statement, CREATE PROCEDURE. Native SQL procedures do not require any other program preparation, such as precompiling, compiling, or link-editing source code. Native SQL procedures are executed as SQL statements that are bound in a Db2 package. Native SQL procedures do not have an associated external application program. Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

An *external SQL procedure* is a procedure whose body is written entirely in SQL. The body is written in the SQL procedural language (SQL PL). However, an external SQL procedure is created, implemented, and executed like other external stored procedures.

Procedure

To migrate an external SQL procedure to a native SQL procedure, complete the following steps:

1. Find and save the existing CREATE PROCEDURE and GRANT EXECUTE statements for the existing external SQL procedure.
2. Drop the existing external SQL procedure by using the DROP PROCEDURE statement.
3. Re-create the procedure as a native SQL procedure by using the same CREATE PROCEDURE statement that you used to originally create the procedure, with both of the following changes:
   - If the procedure was defined with the options FENCED or EXTERNAL, remove these keywords.
   - Either remove the WLM ENVIRONMENT keyword, or add the FOR DEBUG MODE clause.
   - If the procedure body contains statements with unqualified names that could refer to either a column or an SQL variable or parameter, qualify these names. Otherwise, you might need to change the statement.

     Db2 resolves these names differently depending on whether the procedure is an external SQL procedure or a native SQL procedure. For external SQL procedures, Db2 first treats the name as a variable or parameter if one exists with that name. For native SQL procedures, Db2 first treats the name as a column if a column exists with that name. For example, consider the following statement:

     ```
     CREATE PROCEDURE P1 (INOUT C1 INT) ... SELECT C1 INTO xx FROM T1
     ```

In the preceding example, if P1 is an external SQL procedure, C1 is a parameter. For native SQL procedures, C1 is a column in table T1. If such a column does not exist, C1 is a parameter.

4. Issue the same GRANT EXECUTE statements that you used to originally grant privileges for this stored procedure.
5. Increase the value of the TIME parameter on the job statement for applications that call stored procedures.

   Important: This change is necessary because time for SQL external stored procedures is charged to the WLM address space, while time for native SQL stored procedures is charged to the address space of the task.
6. Test your new native SQL procedure.

Related tasks

Creating native SQL procedures
A *native SQL procedure* is a procedure whose body is written entirely in SQL and is created by issuing a single SQL statement, CREATE PROCEDURE.

Related reference

CREATE PROCEDURE (SQL - native) (Db2 SQL)

GRANT (function or procedure privileges) (Db2 SQL)

DROP (Db2 SQL)

Using the Db2 precompiler to assist you in converting an external SQL procedure to a native SQL procedure
The Db2 precompiler can be useful when considering any conversion of an external SQL procedure to a native SQL procedure.

About this task

Use the Db2 precompiler to inspect the SQL procedure source from a native SQL PL perspective. A listing is produced that helps to isolate problems and incompatibilities between external and native SQL procedure coding. Source changes can then be made before making any changes in Db2.

Procedure

To inspect the quality of native SQL PL source coding using the Db2 precompiler:

1. Copy the original SQL PL source code to a FB80 data set. Reformat the source as needed to fit within the precompiler margins.
2. Precompile the SQL PL source by executing program DSNHPSM with the HOST(SQLPL) option.
3. Inspect the produced listing (SYSPRINT). Pay attention to error and warning messages.
4. Modify the SQL PL source to address coding problems that are identified by messages in the listing.
5. Repeat steps “1” on page 293 through “4” on page 293 until all error and warning messages are resolved. Address informational messages as needed.
6. Copy the modified SQL PL source file back to its original source format, reformatting as needed.

Results

Sample JCL DSNTEJ67 demonstrates this process for an external SQL procedure that was produced using the Db2 SQL procedure processor DSNTPSMP.
Related reference

Sample programs to help you prepare and run external SQL procedures

Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

Creating an external SQL procedure by using DSNTPSMP

The SQL procedure processor, DSNTPSMP, is one of several methods that you can use to create and prepare an external SQL procedure. DSNTPSMP is a REXX stored procedure that you can invoke from your application program.

Before you begin

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

Set up support for external SQL procedures.

Also ensure that you have the required authorizations, as indicated in the following table, for invoking DSNTPSMP.

Table 57. Required authorizations for invoking DSNTPSMP

Required authorization	**Associated syntax for the authorization**
Procedure privilege to run application programs that invoke the stored procedure.	EXECUTE ON PROCEDURE SYSPROC.DSNTPSMP
Collection privilege to use BIND to create packages in the specified collection. You can use an asterisk (*) as the identifier for a collection.	CREATE ON COLLECTION *collection-id*
Package privilege to use BIND or REBIND to bind packages in the specified collection.	BIND ON PACKAGE *collection-id*.*
System privilege to use BIND with the ADD option to create packages and plans.	BINDADD
Schema privilege to create, alter, or drop stored procedures in the specified schema. The BUILDOWNER authorization ID must have the CREATEIN privilege on the schema. You can use an asterisk (*) as the identifier for a schema.	CREATEIN, ALTERIN, DROPIN ON SCHEMA *schema-name*
Table privileges to select or delete from, insert into, or update the specified catalog tables.	SELECT ON TABLE SYSIBM.SYSROUTINES SELECT ON TABLE SYSIBM.SYSPARMS SELECT, INSERT, UPDATE, DELETE ON TABLE SYSIBM.SYSROUTINES_SRC SELECT, INSERT, UPDATE, DELETE ON TABLE SYSIBM.SYSROUTINES_OPTS ALL ON TABLE SYSIBM.SYSPSMOUT
Any privileges that are required for the SQL statements and that are contained within the SQL procedure body. These privileges must be associated with the OWNER *authorization-id* that is specified in your bind options. The default owner is the user that is invoking DSNTPSMP.	Syntax varies depending on the SQL procedure body

Procedure

To create an external SQL procedure by using DSNTPSMP:

1. Write an application program that calls DSNTPSMP. Include the following items in your program:

   - A CLOB host variable that contains a CREATE PROCEDURE statement for the external SQL procedure. That statement should include the FENCED keyword or the EXTERNAL keyword, and the procedure body, which is written in SQL.

     Alternatively, instead of defining a host variable for the CREATE PROCEDURE statement, you can store the statement in a data set member.

   - An SQL CALL statement with the BUILD function. The CALL statement should use the proper syntax for invoking DSNTPSMP.

     Pass the SQL procedure source to DSNTPSMP as one of the following input parameters:

     SQL-procedure-source
     Use this parameter if you defined a host variable in your application to contain the CREATE PROCEDURE statement.

     source-data-set-name
     Use this parameter if you stored the CREATE PROCEDURE statement in a data set.

   - Based on the return value from the CALL statement, issue either an SQL COMMIT or a ROLLBACK statement. If the return value is 0 or 4, issue a COMMIT statement. Otherwise, issue a ROLLBACK statement.

     You must process the result set before issuing the COMMIT or ROLLBACK statement.

     A QUERYLEVEL request must be followed by the COMMIT statement.

2. Precompile, compile, and link-edit the application program.
3. Bind a package for the application program.
4. Run the application program.

Related concepts

SQL procedure body
The body of an SQL procedure contains one or more SQL statements. In the SQL procedure body, you can also declare and use variables, conditions, return codes, statements, cursors, and handlers.

Related reference

CREATE PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

Db2 for z/OS SQL procedure processor (DSNTPSMP)
The SQL procedure processor, DSNTPSMP, is a REXX stored procedure that you can use to prepare an external SQL procedure for execution.

You can also use DSNTPSMP to perform selected steps in the preparation process or delete an existing external SQL procedure. DSNTPSMP is the only preparation method for enabling external SQL procedures to be debugged with either the SQL Debugger or the Unified Debugger.

DSNTPSMP requires that your system EBCDIC CCSID also be compatible with the C compiler. Using an incompatible CCSID results in compile-time errors. Examples of incompatible CCSIDs include 290, 930, 1026, and 1155. If your system EBCDIC CCSID is not compatible, do not just change it. Contact IBM Support for help.

Sample startup procedure for a WLM address space for DSNTPSMP
You must run DSNTPSMP in a WLM-established stored procedures address space. You should run only DSNTPSMP in that address space, and you must limit the address space to run only one task concurrently.

This example shows how to set up a WLM address space for DSNTPSMP.

Recommendation: Use the core WLM environment DSNWLM_REXX. Job DSNTIJMV creates an address space procedure called DSNWLMR for this environment.

The following example shows sample JCL for a startup procedure for the address space in which DSNTPSMP runs.

```
//DSNWLMR  PROC DB2SSN=DSN,NUMTCB=1,APPLENV=DSNWLM_REXX          1
//*
//WLMTPSMP EXEC PGM=DSNX9WLM,TIME=1440,                       2
//              PARM='&DB2SSN,&NUMTCB,&APPLENV',
//              REGION=0M,DYNAMNBR=10
//STEPLIB  DD  DISP=SHR,DSN=DSN1010.SDSNEXIT                   3
//         DD  DISP=SHR,DSN=DSN1010.SDSNLOAD
//         DD  DISP=SHR,DSN=CBC.SCCNCMP
//         DD  DISP=SHR,DSN=CEE.SCEERUN
//         DD  DISP=SHR,DSN=DSN1010.DBRMLIB.DATA               3
//SYSEXEC  DD  DISP=SHR,DSN=DSN1010.SDSNCLST                   4
//SYSTSPRT DD  SYSOUT=A
//CEEDUMP  DD  SYSOUT=A
//SYSABEND DD  DUMMY
//*
//SQLDBRM  DD  DISP=SHR,DSN=DSN1010.DBRMLIB.DATA               5
//SQLCSRC  DD  DISP=SHR,DSN=DSN1010.SRCLIB.DATA                6
//SQLLMOD  DD  DISP=SHR,DSN=DSN1010.RUNLIB.LOAD                7
//SQLLIBC  DD  DISP=SHR,DSN=CEE.SCEEH.H                       8
//         DD  DISP=SHR,DSN=CEE.SCEEH.SYS.H
//SQLLIBL  DD  DISP=SHR,DSN=CEE.SCEELKED                      9
//         DD  DISP=SHR,DSN=DSN1010.SDSNLOAD
//SYSMSGS  DD  DISP=SHR,DSN=CEE.SCEEMSGP(EDCPMSGE)            10
//*
//* DSNTPSMP Configuration File - CFGTPSMP (optional)         11
//*          A site-provided sequential data set or member, used to
//*         define customized operation of DSNTPSMP in this APPLENV
//*
//* CFGTPSMP DD  DISP=SHR,DSN=
//*
//SQLSRC   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),            12
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLPRINT DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLTERM  DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLOUT   DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLCPRT  DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLUT1   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLUT2   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLCIN   DD  UNIT=SYSALLDA,SPACE=(32000,(20,20))
//SQLLIN   DD  UNIT=SYSALLDA,SPACE=(3200,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSMOD   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLDUMMY DD  DUMMY
```

Notes:

1

APPLENV specifies the application environment in which DSNTPSMP runs. To ensure that DSNTPSMP always uses the correct data sets and parameters for preparing each external SQL procedure, you can set up different application environments for preparing stored procedures with different program preparation requirements. For example, if all payroll applications use the same set of data sets during program preparation, you could set up an application environment called PAYROLL for preparing only payroll applications. The startup procedure for PAYROLL would point to the data sets that are used for payroll applications.

DB2SSN specifies the Db2 subsystem name.

NUMTCB specifies the number of programs that can run concurrently in the address space. You should always set NUMTCB to 1 to ensure that executions of DSNTPSMP occur serially.

2

WLMTPSMP specifies the address space in which DSNTPSMP runs.

DYNAMNBR reserves space for dynamic allocation of data sets during the SQL procedure preparation process.

3
STEPLIB specifies the Db2 load libraries, the z/OS C/C++ compiler library, and the Language Environment run time library that DSNTPSMP uses when it runs. At least one library must not be APF authorized.

4
SYSEXEC specifies the library that contains the REXX exec DSNTPSMP.

5
SQLDBRM is an output data set that specifies the library into which DSNTPSMP puts the DBRM that it generates when it precompiles your external SQL procedure.

6
SQLCSRC is an output data set that specifies the library into which DSNTPSMP puts the C source code that it generates from the external SQL procedure source code. This data set should have a logical record length of 80.

7
SQLLMOD is an output data set that specifies the library into which DSNTPSMP puts the load module that it generates when it compiles and link-edits your external SQL procedure.

8
SQLLIBC specifies the library that contains standard C header files. This library is used during compilation of the generated C program.

9
SQLLIBL specifies the following libraries, which DSNTPSMP uses when it link-edits the external SQL procedure:

- Language Environment link-edit library
- Db2 load library

10
SYSMSGS specifies the library that contains messages that are used by the C prelink-edit utility.

11
CFGTPSMP specifies an optional data set that you can use to customize DSNTPSMP, including specifying the compiler level. For details on all of the options that you can set in this file and how to set them, see the DSNTPSMP CLIST comments.

12
The DD statements that follow describe work file data sets that are used by DSNTPSMP.

Related tasks

Converting from the AMI-based MQ functions to the MQI-based MQ functions (Db2 Installation and Migration)

CALL statement syntax for invoking DSNTPSMP
You can invoke the SQL procedure processor, DSNTPSMP, from an application program by using an SQL CALL statement. DSNTPSMP prepares an external SQL procedure.

The following diagrams show the syntax of invoking DSNTPSMP through the SQL CALL statement:

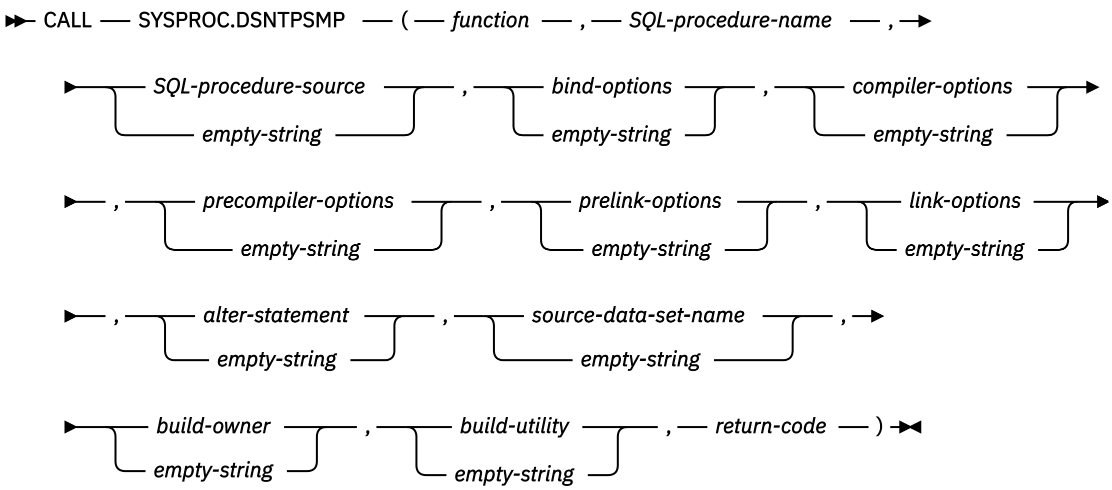


Figure 16. DSNTPSMP syntax

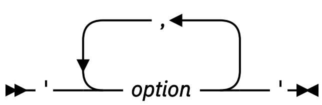


Figure 17. CALL DSNTPSMP bind-options, compiler-options, precompiler-options, prelink-options, link-options

Note: You must specify:

- The DSNTPSMP parameters in the order listed
- The empty string if an optional parameter is not required for the function
- The options in the order: bind, compiler, precompiler, prelink, and link

The DSNTPSMP parameters are:

function
A VARCHAR(20) input parameter that identifies the task that you want DSNTPSMP to perform. The tasks are:

BUILD
Creates the following objects for an external SQL procedure:

- A DBRM, in the data set that DD name SQLDBRM points to
- A load module, in the data set that DD name SQLLMOD points to
- The C language source code for the external SQL procedure, in the data set that DD name SQLCSRC points to
- The stored procedure package
- The stored procedure definition

The following input parameters are required for the BUILD function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

If you choose the BUILD function, and an external SQL procedure with name *SQL-procedure-name* already exists, DSNTPSMP issues an error message and terminates.

BUILD_DEBUG
Creates the following objects for an external SQL procedure and includes the preparation necessary to debug the external SQL procedure with the SQL Debugger and the Unified Debugger:

- A DBRM, in the data set that DD name SQLDBRM points to
- A load module, in the data set that DD name SQLLMOD points to
- The C language source code for theexternal SQL procedure, in the data set that DD name SQLCSRC points to
- The stored procedure package
- The stored procedure definition

The following input parameters are required for the BUILD_DEBUG function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

If you choose the BUILD_DEBUG function, and an external SQL procedure with name *SQL-procedure-name* already exists, DSNTPSMP issues an error message and terminates.

REBUILD
Replaces all objects that were created by the BUILD function for an external SQL procedure, if it exists, otherwise creates those objects.

The following input parameters are required for the REBUILD function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

REBUILD_DEBUG
Replaces all objects that were created by the BUILD_DEBUG function for an external SQL procedure, if it exists, otherwise creates those objects, and includes the preparation necessary to debug the external SQL procedure with the SQL Debugger and the Unified Debugger.

The following input parameters are required for the REBUILD_DEBUG function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

REBIND
Binds the external SQL procedure package for an existing external SQL procedure.

The following input parameter is required for the REBIND function:

SQL-procedure name

DESTROY
Deletes the following objects for an existing external SQL procedure:

- The DBRM, from the data set that DD name SQLDBRM points to
- The load module, from the data set that DD name SQLLMOD points to
- The C language source code for the external SQL procedure, from the data set that DD name SQLCSRC points to
- The stored procedure package
- The stored procedure definition

The following input parameter is required for the DESTROY function:

SQL-procedure name

ALTER
Updates the registration for an existing external SQL procedure.

The following input parameters are required for the ALTER function:

SQL-procedure name
alter-statement

ALTER_REBUILD
Updates an existing external SQL procedure.

The following input parameters are required for the ALTER_REBUILD function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

ALTER_REBUILD_DEBUG
Updates an existing external SQL procedure, and includes the preparation necessary to debug the external SQL procedure with the SQL Debugger and the Unified Debugger.

The following input parameters are required for the ALTER_REBUILD_DEBUG function:

SQL-procedure name
SQL-procedure-source or *source-data-set-name*

ALTER_REBIND
Updates the registration and binds the SQL package for an existing external SQL procedure.

The following input parameters are required for the ALTER_REBIND function:

SQL-procedure name
alter-statement

QUERYLEVEL
Obtains the interface level of the build utility invoked. No other input is required.

SQL-procedure-name
A VARCHAR(261) input parameter that specifies the external SQL procedure name.

The name can be qualified or unqualified. The name must match the procedure name that is specified within the CREATE PROCEDURE statement that is provided in *SQL-procedure-source* or that is obtained from *source-data-set-name*. In addition, the name must match the procedure name that is specified within the ALTER PROCEDURE statement that is provided in *alter-statement*. Do not mix qualified and unqualified references.

SQL-procedure-source
A CLOB(2M) input parameter that contains the CREATE PROCEDURE statement for the external SQL procedure. If you specify an empty string for this parameter, you need to specify the name *source-data-set-name* of a data set that contains the external SQL procedure source code.

bind-options
A VARCHAR(1024) input parameter that contains the options that you want to specify for binding the external SQL procedure package. Do not specify the MEMBER or LIBRARY option for the Db2 BIND PACKAGE command.

compiler-options
A VARCHAR(255) input parameter that contains the options that you want to specify for compiling the C language program that Db2 generates for the external SQL procedure.

precompiler-options
A VARCHAR(255) input parameter that contains the options that you want to specify for precompiling the C language program that Db2 generates for the external SQL procedure. Do not specify the HOST option.

prelink-options
A VARCHAR(255) input parameter that contains the options that you want to specify for prelinking the C language program that Db2 generates for the external SQL procedure.

link-options
A VARCHAR(255) input parameter that contains the options that you want to specify for linking the C language program that Db2 generates for the external SQL procedure.

alter-statement
A VARCHAR(32672) input parameter that contains the SQL ALTER PROCEDURE statement to process with the ALTER or ALTER_REBIND function.

source-data-set-name
A VARCHAR(80) input parameter that contains the name of a z/OS sequential data set or partitioned data set member that contains the source code for the external SQL procedure. If you specify an empty string for this parameter, you need to provide the external SQL procedure source code in *SQL-procedure-source*.

build-owner
A VARCHAR(130) input parameter that contains the SQL identifier to serve as the build owner for newly created SQL stored procedures.

When this parameter is not specified, the value defaults to the value in the CURRENT SQLID special register when the build utility is invoked.

build-utility
A VARCHAR(255) input parameter that contains the name of the build utility that is invoked. The qualified form of the name is suggested, for example, SYSPROC.DSNTPSMP.

return-code
A VARCHAR(255) output parameter in which Db2 puts the return code from the DSNTPSMP invocation. The values are:

0
Successful invocation. The calling application can optionally retrieve the result set and then issue the required SQL COMMIT statement.

4
Successful invocation, but warnings occurred. The calling application should retrieve the warning messages in the result set and then issue the required SQL COMMIT statement.

8
Failed invocation. The calling application should retrieve the error messages in the result set and then issue the required SQL ROLLBACK statement.

99*x*
Where *x* is a digit between 0 and 9. Failed invocation with severe errors. The calling application should retrieve the error messages in the result set and then issue the required SQL ROLLBACK statement. To view error messages that are not in the result set, see the job log of the address space for the DSNTPSMP execution.

999
Unknown severe internal error

998
APF environment setup error

997
DSNREXX setup error

996
Global temporary table setup error

995
Internal REXX programming error

1.2*x*
Where *x* is a digit between 0 and 9. Level of DSNTPSMP when request is QUERYLEVEL. The calling application can retrieve the result set for additional information about the release and service level and then issue the required SQL COMMIT statement.

Related reference
Descriptions of SQL processing options

You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

BIND and REBIND options for packages, plans, and services (Db2 Commands)
Compiler Options (C/C++) (XL C/C++ User's Guide)
Binder options reference (MVS Program Management: User's Guide and Reference)

Examples of invoking the SQL procedure processor (DSNTPSMP)
You can invoke the BUILD, DESTROY, REBUILD, and REBIND functions of DSNTPSMP.

DSNTPSMP BUILD function: Call DSNTPSMP to build an external SQL procedure. The information that DSNTPSMP needs is listed in the following table:

Table 58. The functions DSNTPSMP needs to BUILD an SQL procedure

Function	BUILD
External SQL procedure name	MYSCHEMA.SQLPROC
Source location	String in CLOB host variable procsrc
Bind options	VALIDATE(BIND)
Compiler options	SOURCE, LIST, LONGNAME, RENT
Precompiler options	SOURCE, XREF, STDSQL(NO)
Prelink options	None specified
Link options	AMODE=31, RMODE=ANY, MAP, RENT
Build utility	SYSPROC.DSNTPSMP
Return value	String returned in varying-length host variable returnval

The CALL statement is:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('BUILD','MYSCHEMA.SQLPROC',:procsrc,
 'VALIDATE(BIND)',
 'SOURCE,LIST,LONGNAME,RENT',
 'SOURCE,XREF,STDSQL(NO)',
 '',
 'AMODE=31,RMODE=ANY,MAP,RENT',
 '','','','SYSPROC.DSNTPSMP',
 :returnval);
```

DSNTPSMP DESTROY function: Call DSNTPSMP to delete an external SQL procedure definition and the associated load module. The information that DSNTPMSP needs is listed in the following table:

Table 59. The functions DSNTPSMP needs to DESTROY an SQL procedure

Function	DESTROY
External SQL procedure name	MYSCHEMA.OLDPROC
Return value	String returned in varying-length host variable returnval

The CALL statement is:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('DESTROY','MYSCHEMA.OLDPROC','',
 '','','','','',
 '','','','',
 :returnval);
```

DSNTPSMP REBUILD function: Call DSNTPSMP to re-create an existing external SQL procedure. The information that DSNTPMSP needs is listed in the following table:

Table 60. The functions DSNTPSMP needs to REBUILD an SQL procedure

Function	**REBUILD**
External SQL procedure name	MYSCHEMA.SQLPROC
Bind options	VALIDATE(BIND)
Compiler options	SOURCE, LIST, LONGNAME, RENT
Precompiler options	SOURCE, XREF, STDSQL(NO)
Prelink options	None specified
Link options	AMODE=31, RMODE=ANY, MAP, RENT
Source data set name	Member PROCSRC of partitioned data set DSN1110.SDSNSAMP
Return value	String returned in varying-length host variable returnval

The CALL statement is:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBUILD','MYSCHEMA.SQLPROC','',
 'VALIDATE(BIND)',
 'SOURCE,LIST,LONGNAME,RENT',
 'SOURCE,XREF,STDSQL(NO)',
 '',
 'AMODE=31,RMODE=ANY,MAP,RENT',
 '','DSN1110.SDSNSAMP(PROCSRC)','','',
 :returnval);
```

If you want to re-create an existing external SQL procedure for debugging with the SQL Debugger and the Unified Debugger, use the following CALL statement, which includes the REBUILD_DEBUG function:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBUILD_DEBUG','MYSCHEMA.SQLPROC','',
 'VALIDATE(BIND)',
 'SOURCE,LIST,LONGNAME,RENT',
 'SOURCE,XREF,STDSQL(NO)',
 '',
 'AMODE=31,RMODE=ANY,MAP,RENT',
 '','DSN1110.SDSNSAMP(PROCSRC)','','',
 :returnval);
```

DSNTPSMP REBIND function: Call DSNTPSMP to rebind the package for an existing external SQL procedure. The information that DSNTPMSP needs is listed in the following table:

Table 61. The functions DSNTPSMP needs to REBIND an SQL procedure

Function	**REBIND**
ExternalSQL procedure name	MYSCHEMA.SQLPROC
Bind options	VALIDATE(RUN), ISOLATION(RR)
Return value	String returned in varying-length host variable returnval

The CALL statement is:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBIND','MYSCHEMA.SQLPROC','',
 'VALIDATE(RUN),ISOLATION(RR)','','','','',
 '','','','',
 :returnval);
```

Result set that the SQL procedure processor (DSNTPSMP) returns
DSNTPSMP returns one result set that contains messages and listings. You can write your client program to retrieve information from this result set. Because DSNTPSMP is a stored procedure, use the same technique that you would use to write a program to receive result sets from any stored procedure.

Each row of the result set contains the following information:

Processing step
The step in the DSNTPSMP *function* process to which the message applies.

DD name
The DD statement that identifies the data set that contains the message.

Sequence number
The sequence number of a line of message text within a message.

Message
A line of message text.

Rows in the message result set are ordered by processing step, DD name, and sequence number.

For an example of how to process a result set from DSNTPSMP, see the Db2 sample program DSNTEJ65.

Related concepts

Db2 for z/OS SQL procedure processor (DSNTPSMP)
The SQL procedure processor, DSNTPSMP, is a REXX stored procedure that you can use to prepare an external SQL procedure for execution.

Job DSNTEJ65 (Db2 Installation and Migration)

Related tasks

Writing a program to receive the result sets from a stored procedure
You can write a program to receive results set from a stored procedure for either a fixed number of result sets, for which you know the contents, or a variable number of result sets, for which you do not know the contents.

Creating an external SQL procedure by using JCL

Using JCL is one of several ways that you can create and prepare an external SQL procedure.

Before you begin

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

About this task

Restriction: You cannot use JCL to prepare an external SQL procedure for debugging with the Db2 stored procedure debugger or the Unified Debugger. If you plan to use either of these debugging tools, use either DSNTPSMP or IBM Data Studio to create the external SQL procedure.

Procedure

To create an external SQL procedure by using JCL, include the following job steps in your JCL job:

1. Issue a CREATE PROCEDURE statement that includes either the FENCED keyword or the EXTERNAL keyword and the procedure body, which is written in SQL.

   Alternatively, you can issue the CREATE PROCEDURE statement dynamically by using an application such as SPUFI, DSNTEP2, DSNTIAD, or the command line processor.

   Tip: If the routine body of the CREATE PROCEDURE statement contains embedded semicolons, change the default SQL terminator character from a semicolon to some other special character, such as the percent sign (%).

This statement defines the stored procedure to Db2. Db2 stores the definition in the Db2 catalog.

2. Run program DSNHPC with the HOST(SQL) option.

   This program converts the external SQL procedure source statements into a C language program. DSNHPC also writes a new CREATE PROCEDURE statement in the data set that is specified in the SYSUT1 DD statement.

3. Precompile, compile, and link-edit the generated C program by using one of the following techniques:

   - The Db2 precompiler and JCL instructions to compile and link-edit the program
   - The SQL statement coprocessor

   When you perform this step, specify the following settings:

   - Give the DBRM the same name as the name of the load module for the external SQL procedure.
   - Specify MARGINS(1,80) for the MARGINS SQL processing option.
   - Specify the NOSEQ compiler option.

   This process produces an executable C language program.

4. Bind the resulting DBRM into a package.

Example

Suppose that you define an external SQL procedure by issuing the following CREATE PROCEDURE statement dynamically:

```
CREATE PROCEDURE DEVL7083.EMPDTLSS
(
 IN  PEMPNO        CHAR(6)
,OUT PFIRSTNME     VARCHAR(12)
,OUT PMIDINIT      CHAR(1)
,OUT PLASTNAME     VARCHAR(15)
,OUT PWORKDEPT     CHAR(3)
,OUT PHIREDATE     DATE
,OUT PSALARY       DEC(9,2)
,OUT PSQLCODE      INTEGER
)
RESULT SETS 0
MODIFIES SQL DATA
FENCED
NO DBINFO
WLM ENVIRONMENT DB2AWLMR
STAY RESIDENT NO
COLLID DEVL7083
PROGRAM TYPE MAIN
RUN OPTIONS 'TRAP(OFF),RPTOPTS(OFF)'
COMMIT ON RETURN NO
LANGUAGE SQL
BEGIN
DECLARE SQLCODE INTEGER;
DECLARE SQLSTATE CHAR(5);
DECLARE EXIT HANDLER FOR SQLEXCEPTION SET PSQLCODE = SQLCODE;
SELECT
       FIRSTNME
     , MIDINIT
     , LASTNAME
     , WORKDEPT
     , HIREDATE
     , SALARY
INTO   PFIRSTNME
     , PMIDINIT
     , PLASTNAME
     , PWORKDEPT
     , PHIREDATE
     , PSALARY
FROM   EMP
WHERE  EMPNO = PEMPNO
;
END
```

You can use JCL that is similar to the following JCL to prepare the procedure:

```
//ADMF001S JOB (999,POK),'SQL C/L/B/E',CLASS=A,MSGCLASS=T,
// NOTIFY=ADMF001,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB2AU.PROCLIB)
//*
//JOBLIB  DD  DSN=DB2A.SDSNEXIT,DISP=SHR
//        DD  DSN=DB2A.SDSNLOAD,DISP=SHR
//        DD  DSN=CEE.SCEERUN,DISP=SHR
//*----------------------------------------------------------
//*        STEP 01: PRECOMP, COMP, LKED AN SQL PROCEDURE
//*----------------------------------------------------------
//SQL01 EXEC DSNHSQL,MEM=EMPDTLSS,
//  PARM.PC='HOST(SQL),SOURCE,XREF,MAR(1,80),STDSQL(NO)',
//  PARM.PCC='HOST(C),SOURCE,XREF,MAR(1,80),STDSQL(NO),TWOPASS',
//  PARM.C='SOURCE LIST MAR(1,80) NOSEQ LO RENT',
//              PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT'
//PC.SYSLIB DD DUMMY
//PC.SYSUT2 DD DSN=&&SPDML,DISP=(,PASS), &lt;=MAKE IT PERMANENT, IF YOU
//            UNIT=SYSDA,SPACE=(TRK,1),      WANT TO USE IT LATER
//            DCB=(RECFM=FB,LRECL=80)
//PC.SYSIN     DD DISP=SHR,DSN=SG247083.PROD.DDL(&MEM.)
//PC.SYSCIN    DD DISP=SHR,DSN=SG247083.TEST.C.SOURCE(&MEM.)
//PCC.SYSIN    DD DISP=SHR,DSN=SG247083.TEST.C.SOURCE(&MEM.)
//PCC.SYSLIB   DD DUMMY
//PCC.DBRMLIB  DD DISP=SHR,DSN=SG247083.DEVL.DBRM(&MEM.)
//LKED.SYSLMOD DD DISP=SHR,DSN=SG247083.DEVL.LOAD(&MEM.)
//LKED.SYSIN DD *     INCLUDE SYSLIB(DSNRLI)     NAME EMPDTLSS(R)
/*
//*----------------------------------------------------------
//*        STEP 02: BIND THE PROGRAM
//*----------------------------------------------------------
//SQL02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD DSN=SG247083.DEVL.DBRM,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//REPORT   DD SYSOUT=*
//SYSIN    DD *
//SYSTSIN DD *
 DSN SYSTEM(DB2A)
 BIND PACKAGE(DEVL7083) MEMBER(EMPDTLSS) VALIDATE(BIND) -
 OWNER(DEVL7083)
 END
//*
```

Related concepts

SQL procedure body
The body of an SQL procedure contains one or more SQL statements. In the SQL procedure body, you can also declare and use variables, conditions, return codes, statements, cursors, and handlers.

The Db2 command line processor (Db2 Commands)

Related tasks

Changing SPUFI defaults
Before you execute SQL statements in SPUFI, you can change the default execution behavior, such as the SQL terminator and the isolation level.

Creating an external SQL procedure by using DSNTPSMP
The SQL procedure processor, DSNTPSMP, is one of several methods that you can use to create and prepare an external SQL procedure. DSNTPSMP is a REXX stored procedure that you can invoke from your application program.

Developing database routines (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio)

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

DSNTEP2 and DSNTEP4 sample programs

You can use the DSNTEP2 or DSNTEP4 programs to execute SQL statements dynamically.

DSNTIAD sample program
You can use the DSNTIAD program to execute dynamic SQL statements other than SELECT statements.

BIND PACKAGE (DSN) (Db2 Commands)
CREATE PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

Sample programs to help you prepare and run external SQL procedures

Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

See the prolog of each sample for specific instructions.

The following table lists the sample jobs that Db2 provides for external SQL procedures.

Table 62. External SQL procedure samples shipped with Db2

Member that contains source code	Contents	Purpose
DSNHSQL	JCL procedure	Precompiles, compiles, prelink-edits, and link-edits an external SQL procedure
DSNTEJ63	JCL job	Invokes JCL procedure DSNHSQL to prepare external SQL procedure DSN8ES1 for execution
DSN8ES1	External SQL procedure	A stored procedure that accepts a department number as input and returns a result set that contains salary information for each employee in that department
DSNTEJ64	JCL job	Prepares client program DSN8ED3 for execution
DSN8ED3	C program	Calls SQL procedure DSN8ES1
DSN8ES2	External SQL procedure	A stored procedure that accepts one input parameter and returns two output parameters. The input parameter specifies a bonus to be awarded to managers. The external SQL procedure updates the BONUS column of DSN1110.SDSNSAMP. If no SQL error occurs when the external SQL procedure runs, the first output parameter contains the total of all bonuses awarded to managers and the second output parameter contains a null value. If an SQL error occurs, the second output parameter contains an SQLCODE.
DSN8ED4	C program	Calls the SQL procedure processor, DSNTPSMP, to prepare DSN8ES2 for execution
DSN8WLMP	JCL procedure	A sample startup procedure for the WLM-established stored procedures address space in which DSNTPSMP runs
DSN8ED5	C program	Calls external SQL procedure DSN8ES2

Member that contains source code	Contents	Purpose
DSNTEJ65	JCL job	Prepares and executes programs DSN8ED4 and DSN8ED5. DSNTEJ65 uses DSNTPSMP, the SQL procedure processor, which requires that the default EBCDIC CCSID that is used by Db2 also be compatible with the C compiler. Do not run DSNTEJ65 if the default EBCDIC CCSID for Db2 is not compatible with the C compiler. Examples of incompatible CCSIDs include 290, 930, 1026, and 1155.
DSNTEJ67	JCL job	Prepares an existing external SQL procedure (sample DSN8.DSN8ES2) for conversion to a native SQL procedure. DSNTEJ67 obtains the source of external SQL procedure DSN8.DSN8ES2 from the catalog and formats it into a data set. DSNTEJ67 executes DSNHPSM with HOST(SQLPL), obtains a listing for the source, and replaces the offending procedure options in the source data set.
DSNTIJRT	JCL job	Prepares a Db2 for z/OS server for operation with the SQL Debugger and the Unified Debugger

Table 62. External SQL procedure samples shipped with Db2 (continued)

DSN8ED4
Demonstrates how to use an application program to call DSNTPSMP, the Db2 SQL Procedures Processor.

```
/********************************************************************* 00010000
* Module name = DSN8ED4 (sample program)                             * 00020000
*                                                                    * 00030000
* DESCRIPTIVE NAME: Sample client for:                               * 00040000
*                  DSNTPSMP (DB2 SQL Procedures Processor)           * 00050000
*                                                                    * 00060000
*     LICENSED MATERIALS - PROPERTY OF IBM                           * 00070000
*     5625-DB2                                                       * 00080000
*     (C) COPYRIGHT 1982, 2003 IBM CORP.  ALL RIGHTS RESERVED.       * 00090000
*                                                                    * 00100000
*     STATUS = VERSION 8                                             * 00110000
*                                                                    * 00120000
* Function: Demonstrates how to use an application program to call   * 00130000
*          DSNTPSMP, the DB2 SQL Procedures Processor.  DSN8ED4      * 00140000
*          collects and passes user-provided SQL Procedure source    * 00150000
*          code and prep options to DSNTPSMP, and outputs the        * 00160000
*          report(s), if any, returned from DSNTPSMP by result set. * 00170000
*                                                                    * 00180000
* Notes:                                                             * 00190000
*   Dependencies: Requires SYSPROC.DSNTPSMP                          * 00200000
*                                                                    * 00210000
*   Restrictions:                                                    * 00220000
*                                                                    * 00230000
* Module type: C program                                             * 00240000
*   Processor: DB2 Precompiler                                       * 00250000
*              IBM C/C++ for OS/390 V1R3 or higher                   * 00260000
* Module size: See linkedit output                                   * 00270000
*  Attributes: Reentrant and reusable                                * 00280000
*                                                                    * 00290000
* Entry point: DSN8ED4                                               * 00300000
*     Purpose: See Function                                          * 00310000
*     Linkage: Standard MVS program invocation, three parameters.    * 00320000
*                                                                    * 00330000
*  Parameters: DSN8ED4 uses the C "main" argument convention of      * 00340000
*              argv (argument vector) and argc (argument count).     * 00350000
*                                                                    * 00360000
*              - ARGV[0]: (input) pointer to a char[9],              * 00370000
*                         null-terminated string having the name of  * 00380000
*                         this program (DSN8ED4)                     * 00390000
*              - ARGV[1]: (input) pointer to a char[21],             * 00400000
*                         null-terminated string having the action   * 00410000
```

```
 *                       that DSNTPSMP is to perform:              * 00420000
 *                       - BUILD: Prepare a new SQL Procedure       * 00430000
 *                       - REBUILD: Prepare an existing SQL         * 00440000
 *                       - QUERYLEVEL: Verify DSNTPSMP level     @05* 00450004
 *                       - DESTROY: Remove an SQL Procedure         * 00460000
 *                       - REBIND: Rebind the package of an exist-  * 00470000
 *                           ing SQL Procedure                      * 00480000
 *             - ARGV[2]: (input) pointer to a char[262],           * 00490000
 *                       null-terminated string having the schema   * 00500000
 *                       and name of the SQL Procedure to be        * 00510000
 *                       processed by DSNTPSMP (e.g. DSN8.DSN8ES2)  * 00520000
 *             - ARGV[3]: (input) pointer to a char[9],             * 00530000
 *                       null-terminated string having the author-  * 00540000
 *                       ization id to be used for BUILDOWNER and   * 00550003
 *                       for calling DSNTPSMP.                      * 00560003
 *             - ARGV[4]: (input) pointer to a char[17],            * 00570000
 *                       null-terminated string having the name of  * 00580000
 *                       the server where DSNTPSMP is to be run.    * 00590000
 *                       This is an optional parameter; the local   * 00600000
 *                       server is used if no argument is provided. * 00610000
 *                                                                  * 00620000
 *      Inputs: DSN8ED4 allocates these input DDs:                  * 00630000
 *             - PCOPTS  : Options for the DB2 precompiler          * 00640000
 *             - COPTS   : Options for the C compiler               * 00650000
 *             - PLKDOPTS: Options for the pre-link editor          * 00660000
 *             - LKEDOPTS: Options for the link editor              * 00670000
 *             - BINDOPTS: Options for the DB2 BIND                 * 00680000
 *             - SQLIN   : Source code for the SQL Procedure        * 00690000
 *                                                                  * 00700000
 *     Outputs: DSN8ED4 allocates these output DD                   * 00710000
 *             - REPORT01: First report data set from DSNTPSMP      * 00720000
 *             - REPORT02: Second report data set from DSNTPSMP     * 00730000
 *             - REPORT03: Third report data set from DSNTPSMP      * 00740000
 *                                                                  * 00750000
 * Normal Exit: Return Code: 0000                                   * 00760000
 *             - Message: DSNTPSMP has completed with return code 0  * 00770000
 *             - Message: SQL changes have been committed        @04* 00780000
 *                                                                  * 00790000
 * Normal with Warnings Exit: Return Code: 0004                 +@04* 00800000
 *             - Message: DSNTPSMP has completed with return code 4  * 00810000
 *             - Message: SQL changes have been committed       -@04* 00820000
 *                                                                  * 00830000
 *  Error Exit: Return Code: 0012                                   * 00840000
 *             - Message: DSNTPSMP has completed with return code <n>* 00850000
 *             - Message: The length of the argument specified for   * 00860000
 *                       the <parameter-name> does not fall within  * 00870000
 *                       the required bounds of <minimum-length>    * 00880000
 *                       and <maximum-length>                       * 00890000
 *             - Message: The argument specified for the action     * 00900000
 *                       parameter is invalid                       * 00910000
 *             - Message: Invalid sequence number <sequence-number>  * 00920000
 *                       specified for REPORTnn DD                  * 00930000
 *             - Message: DSN8ED4 was invoked with <parameter-count> * 00940000
 *                       parameters.  At least 3 parameters are     * 00950000
 *                       required                                   * 00960000
 *             - Message: Unable to open <DD-name>                  * 00970000
 *             - Message: Unable to close <DD-name>                 * 00980000
 *             - Message: <formatted SQL text from DSNTIAR>         * 00990000
 *             - Message: SQL changes have been rolled back      @04* 01000000
 *                                                                  * 01010000
 * External References:                                             * 01020000
 *             - Routines/Services: DSNTIAR: DB2 msg text formatter  * 01030000
 *             - Data areas       : None                            * 01040000
 *             - Control blocks   : None                            * 01050000
 *                                                                  * 01060000
 *                                                                  * 01070000
 * Pseudocode:                                                      * 01080000
 *   DSN8ED4:                                                       * 01090000
 *     - call getCallParms to receive and validate call parm arguments* 01100000
 *     - case action                                                * 01110000
 *       - when BUILD, call getReBuildData                          * 01120000
 *       - when DESTROY, call getDestroyData                        * 01130000
 *       - when REBUILD, call getReBuildData                        * 01140000
 *       - when REBIND, call getRebindData                          * 01150000
 *       - when QUERYLEVEL, call getLevelData                       * 01160003
 *       - otherwise call issueInvalidActionError                   * 01170000
 *     - call connectToLocation                                     * 01180000
 *     - call setAuthID to set the current authorization id  @pq53353 * 01190003
 *     - call callDSNTPSMP to invoke the DB2 SQL Procedures Processor * 01200000
 *     - call processDSNTPSMPresultSet to write reports from DSNTPSMP * 01210000
 *     - If no errors, call processSqlCommit to commit work       @04 * 01220000
 *       Else call processSqlRollback to undo work                @04 * 01230000
```

```
*   End DSN8ED4                                                      *
*                                                                    *
*                                                                    *
*  Change activity =                                                 *
*    PQ46962 03/28/2001 changed line feed character to hex 25    @01 *
*    PQ43444 04/12/2001 Disable LEOPTS DD (LE options are not    @02 *
*                       processed by DSNTPSMP).  Remission the   @02 *
*                       leOptions hostvar as alterStmt.          @02 *
*    PQ56601 03/06/2002 Trim +/- continuation characters from    @03 *
*                       BIND options to prevent BIND errors.     @03 *
*                       These characters are often used to con-  @03 *
*                       tinue BIND statements being processed    @03 *
*                       by the DB2 DSN command processor (which  @03 *
*                       uses TSO i/o services that recognize     @03 *
*                       them as continuation characters) but     @03 *
*                       they are not otherwise valid in DB2      @03 *
*                       commands.                                @03 *
*    PQ61782 07/16/2002 Distinguish between DSNTPSMP return code @04 *
*                       and DSN8ED4 return code; Issue SQL COMMIT@04 *
*                       when DSNTPSMP returns rc = 0 or rc = 4;  @04 *
*                       Otherwise issue SQL ROLLBACK             @04 *
*    D55199  12/08/2003 Adjust to use DSNTPSMP 1.2x interface    @05 *
*    D56462  02/12/2004 Allocate maximum of 6 output reports     @06 *
*********************************************************************/

/********************** C library definitions ***********************/
#include        <errno.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>

/**************************** Constants *****************************/
#define         NULLCHAR         '\0' /* Null character            */
#define         RETNRM             0  /* Normal  return code     @04*/
#define         RETWRN             4  /* Warning return code       */
#define         RETERR             8  /* Error return code         */
#define         RETSEV            12  /* Severe error return code  */
#define         INTERFACE       "1.2" /* DSNTPSMP innterface level */

enum flag                  {No, Yes}; /* Settings for flags        */


/**************** Input: SQL Procedure Source Code *****************/
FILE            *sqlInFile;           /* Pointer to SQL source DD  */


/*********** Output: DB2 SQL Procedures Processor Reports ***********/
FILE            *reportDD;            /* Pointer to curr report DD */
char            reportDDName[12];     /* For generated DD name     */
unsigned short  reportLRECL;          /* length req'd for output rec*/


/*********************** Working variables ************************/
unsigned short  resultSetReturned = 0;/* DSNTPSMP result set stat@04*/
long int        DSNTPSMP_rc     = -1; /* DSNTPSMP return code    @04*/
long int        rc               = 0; /* program return code       */
char            levelquery      ='N'; /* Is this a level check?  @05*/


/******************** DB2 SQL Communication Area *******************/
EXEC SQL INCLUDE SQLCA;


/*********************** DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;

  char            authID[9];          /* Authorization id-BUILDOWNER*/

  char            locationName[17];   /* Server location name      */

  char            action[21];         /* Command for PSM processor */
  char            routineName[262];   /* SQL Procedure schema.name */
  SQL TYPE IS CLOB(2M) sqlSource;     /* SQL Procedure source   @05*/

  char            precompOptions[256];/* precompiler options       */
  char            compileOptions[256];/* compilation parameters    */
  char            prelinkOptions[256];/* prelink options           */
  char            linkOptions[256];   /* link-edit options         */
  char            bindOptions[1025];  /* DB2 bind options          */
  char            alterStmt[32672];   /* ALTER PROC text        @02*/

  char            sqlSourceDsn[81];   /* Source data set name      */
```

```
    char            outputString[256];  /* DSNTPSMP status area       */ 02060000
                                                                         02070000
    char            DSNTPSMP_pname[19]  /* DSNTPSMP procedure-name    */ 02080000
                    = "SYSPROC.DSNTPSMP\0";                              02090000
                                                                         02100000
    char            stepName[17];       /* DSNTPSMP stepname          */ 02110000
    char            fileName[9];        /* DSNTPSMP output DD name    */ 02120000
    long int        reportLineNumber;   /* DSNTPSMP report line no.   */ 02130000
    char            reportLine[256];    /* DSNTPSMP report line       */ 02140000
                                                                         02150000
  EXEC SQL END DECLARE SECTION;                                          02160000
                                                                         02170000
                                                                         02180000
  /*************** DB2 Result Set Locator Host Variables **************/ 02190000
  EXEC SQL BEGIN DECLARE SECTION;                                        02200000
    static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSNTPSMP_rs_loc1;    02210000
  EXEC SQL END DECLARE SECTION;                                          02220000
                                                                         02230000
                                                                         02240000
  /********************** DSN8ED4 Function Models *********************/ 02250000
  int main                              /* DSN8ED4 driver             */ 02260000
    ( int argc,                         /* - Input argument count     */ 02270000
      char *argv[]                      /* - Input argument vector    */ 02280000
    );                                                                   02290000
  void getCallParms                     /* Process args to call parms */ 02300000
    ( int argc,                         /* - Input argument count     */ 02310000
      char *argv[]                      /* - Input argument vector    */ 02320000
    );                                                                   02330000
  void getReBuildData( void );          /* Get SQL Proc re/build data */ 02340000
  void getDestroyData( void );          /* Get SQL Proc destroy data  */ 02350000
  void getRebindData( void );           /* Get SQL Proc rebind data   */ 02360000
  void getLevelData( void );            /* Get DSNTPSMP level data    */ 02370003
  void getOptions                       /* Read specified options file*/ 02380000
    ( char *options,                    /* -out: list of options read */ 02390000
      int maxBytes,                     /* - in: max size of list     */ 02400000
      char *optionsDDname               /* - in: name of DD to read   */ 02410000
    );                                                                   02420000
  void getSqlSource( void );            /* Read SQL Procedure Source  */ 02430000
  void setAuthID( void );               /* Set the current DB2 auth id*/ 02440003
  void connectToLocation( void );       /* Connect to DB2 location    */ 02450000
  void callDSNTPSMP( void );            /* Run SQL Procedure Processor*/ 02460000
  void listDSNTPSMPcallParms( void );   /* List parms sent to DSNTPSMP*/ 02470000
  void processDSNTPSMPresultSet( void );/* Process DSNTPSMP rslt sets */ 02480000
  void associateResultSetLocator(void); /* Assoc DSNTPSMP RS locator  */ 02490000
  void allocateResultSetCursor( void ); /* Alloc DSNTPSMP RS cursor   */ 02500000
  void writeDSNTPSMPreports( void );    /* Output a DSNTPSMP report   */ 02510000
  void fetchFromResultSetCursor( void );/* Read DSNTSPMP RS cursor    */ 02520000
  void openReportDataSet                /* Alloc DD for a report      */ 02530000
    ( short int reportNumber            /* - in: Seqeunce number      */ 02540000
    );                                                                   02550000
  void closeReportDataSet( void );      /* Dealloc DD for a report    */ 02560000
  void trimTrailingBlanks               /* Strip off trailing blanks  */ 02570000
    ( char            *string           /* - in: string to be trimmed */ 02580000
    );                                                                   02590000
  void stripContinuationCharacter       /* Strip off trailing - or +  */ 02600000
    ( char            *string           /* - in: string to be trimmed */ 02610000
    );                                                           /*@03*/ 02620000
  void processSqlCommit( void );        /* Commit SQL changes      @04*/ 02630000
  void processSqlRollback( void );      /* Rollback SQL changes    @04*/ 02640000
  void issueDataSetClosingError         /* Handler for ds close error */ 02650000
    ( char            *DDname,          /* - in: name of errant DD    */ 02660000
      int             LEerrno           /* - in: LE diagnostic errno  */ 02670000
    );                                                                   02680000
  void issueDataSetOpeningError         /* Handler for ds open error  */ 02690000
    ( char            *DDname,          /* - in: name of errant DD    */ 02700000
      int             LEerrno           /* - in: LE diagnostic errno  */ 02710000
    );                                                                   02720000
  void issueDataSetReadingError         /* Handler for ds read error  */ 02730000
    ( char            *DDname,          /* - in: name of errant DD    */ 02740000
      int             LEerrno           /* - in: LE diagnostic errno  */ 02750000
    );                                                                   02760000
  void issueInvalidCallParmCountError   /* Handler for parm count err */ 02770000
    ( int argc                          /* - in: no. parms received   */ 02780000
    );                                                                   02790000
  void issueInvalidActionError          /* Handler for unknown action */ 02800000
    ( char *action                      /* - in: action specified     */ 02810000
    );                                                                   02820000
  void issueInvalidLevelError           /* Handler for wrong DSNTPSMP */ 02830003
    ( char *level                       /* - in: level encountered    */ 02840003
    );                                                                   02850003
  void issueInvalidDDnumError           /* Handler for unknown DD seq */ 02860000
    ( short          invalidDDnum       /* - in: invalid DD sequ. no. */ 02870000
```

```
  );
void issueInvalidParmLengthError      /* Handler for parm len error */
  ( char *parmName,                   /* - in: identify of parm     */
    int minLength,                    /* - in: min valid length     */
    int maxLength                     /* - in: max valid length     */
  );
void issueSqlError                    /* Handler for SQL error      */
  ( char *locMsg                      /* - in: Call location        */
  );


int main                              /* DSN8ED4 driver             */
  ( int argc,                         /* - Input argument count     */
    char *argv[]                      /* - Input argument vector    */
  )
  /*******************************************************************
  * Main Driver:                                                     *
  * - Gets arguments for call parms                                  *
  * - Gets processing options and data                               *
  * - Connects to remote location, if one was specified              *
  * - Calls the DB2 SQL Procedure Processor, DSNTSPMP                *
  * - Processes any result set(s) returned from DSNTPSMP             *
  *                                                                  *
  *******************************************************************/
  { /*****************************************************************
    * Extract the following information from the call parms:        *
    * (1) DB2 location name where where SQL Procedure is to be built,*
    *     destroyed, rebuilt, rebound, etc.)                        *
    * (2) DB2 SQL Procedure Processor action (Build,Destroy,...)     *
    * (3) Name of SQL Procedure to be built, destroyed, rebound, etc.*
    *****************************************************************/
    getCallParms( argc,argv );

    /*****************************************************************
    * Collect DSNTPSMP parms appropriate for the user-passed action  *
    *****************************************************************/
    if( rc < RETSEV )
      { if( memcmp( action,"BUILD",5 ) == 0 )
          { getReBuildData();
          }
        else if( memcmp( action,"DESTROY",7 ) == 0 )
          { getDestroyData();
          }
        else if( memcmp( action,"REBUILD",7 ) == 0 )
          { getReBuildData();
          }
        else if( memcmp( action,"REBIND",6 ) == 0 )
          { getRebindData();
          }
        else if( memcmp( action,"QUERYLEVEL",10 ) == 0 )
          { getLevelData();
            levelquery='Y';
          }
        else
          { issueInvalidActionError( action );
          }
      }

    /*****************************************************************
    * Connect to location where the SQL Procedure is to be processed *
    *****************************************************************/
    if( rc < RETSEV  &&  strlen(locationName) > 0 )
      connectToLocation();

    /*****************************************************************
    * Set current DB2 authorization id to use when calling DSNTPSMP  *
    *****************************************************************/
    if( rc < RETSEV )                                   /*@pq53353*/
      setAuthID();

    /*****************************************************************
    * Call the PSM processor                                         *
    *****************************************************************/
    if( rc < RETSEV )
      callDSNTPSMP();

    /*****************************************************************
    * Process the result set, if any, from DSNTPSMP                  *
    *****************************************************************/
    if( resultSetReturned )                                  /*@04*/
      processDSNTPSMPresultSet();

```

```
    /*************************************************************+@04**
    * If DSNTPSMP returns either 0 (normal) or 4 (warnings), commit  *
    * the SQL changes; Otherwise, rollback the SQL changes           *
    ******************************************************************/
    if( DSNTPSMP_rc == RETNRM || DSNTPSMP_rc == RETWRN )
      { processSqlCommit();
        if( rc < DSNTPSMP_rc )
          rc = DSNTPSMP_rc;
      }
    else
      { processSqlRollback();
        if( rc < RETSEV )
          rc = RETSEV;
      }                                                         /*-@04*/

    /*******************************************************************
    * Return highest completion code                                 *
    ******************************************************************/
    return( rc );

  } /* end of main */


void getCallParms                     /* Process args to call parms */
  ( int argc,                         /* - Input argument count     */
    char *argv[]                      /* - Input argument vector    */
  )
  /*********************************************************************
  * Verifies that correct call parms have been passed in:            *
  * - Three parameters (action, routine name, and authorization id)  *
  *   require arguments                                              *
  * - The fourth parameter (location name) is optional               *
  ********************************************************************/
  { if( argc < 4  || argc > 5 )
      { issueInvalidCallParmCountError( argc );
      }
    else if( strlen( argv[1] ) < 1  ||  strlen( argv[1] ) > 20 )
      { issueInvalidParmLengthError("DSNTPSMP Action",1,20);
      }
    else if( strlen( argv[2] ) < 1  ||  strlen( argv[2] ) > 261 )
      { issueInvalidParmLengthError("SQL Procedure schema.name",1,261);
      }
    else if( strlen( argv[3] ) < 1  ||  strlen( argv[3] ) > 8 )
      { issueInvalidParmLengthError("Authorization ID",1,8);
      }
    else
      { strcpy( action,      argv[1] );
        strcpy( routineName, argv[2] );
        strcpy( authID, argv[3] );
      }

    if( argc > 4 )
      if( strlen( argv[4] ) < 1  ||  strlen( argv[4] ) > 16 )
        { issueInvalidParmLengthError("Server Location Name",1,16);
        }
      else
        strcpy( locationName,argv[4] );
    else
      locationName[0] = NULLCHAR;

  } /* end of getCallParms */


void getReBuildData( void )           /* Get SQL Proc re/build data */
  /*********************************************************************
  * Collects the prep options and source data needed by DSNTPSMP to  *
  * perform a BUILD or REBUILD operation.                            *
  ********************************************************************/
  {
    /*******************************************************************
    * Get program prep, bind, and runtime options                    *
    ******************************************************************/
    getOptions( precompOptions,255,"PCOPTS" );
    if( rc < RETSEV )
      getOptions( compileOptions,255,"COPTS" );
    if( rc < RETSEV )
      getOptions( prelinkOptions,255,"PLKDOPTS" );
    if( rc < RETSEV )
      getOptions( linkOptions,255,"LKEDOPTS" );
    if( rc < RETSEV )
      getOptions( bindOptions,1024,"BINDOPTS" );
    /* if( rc < RETSEV )                                         @02*/
```

```
    /*   getOptions( LeOptions,254,"LEOPTS" );                  @02*/

    /*****************************************************************
    * Get the source for the SQL procedure to be prepared            *
    *****************************************************************/
    if( rc < RETSEV )
      getSqlSource();
  } /* end of getReBuildData */


void getDestroyData( void )           /* Get SQL Proc destroy data  */
  /*******************************************************************
  * Gets the name of the package to be freed by DSNTPSMP during a    *
  * DESTROY operation.                                               *
  *******************************************************************/
  {
    /*****************************************************************
    * Set program prep and runtime options to NULLCHAR               *
    *****************************************************************/
    sqlSource.length      = 0;                                /*@05*/
    sqlSource.data[0]     = NULLCHAR;                         /*@05*/
    precompOptions[0]     = NULLCHAR;
    compileOptions[0]     = NULLCHAR;
    prelinkOptions[0]     = NULLCHAR;
    linkOptions[0]        = NULLCHAR;
    alterStmt[0]          = NULLCHAR;                         /*@02*/
    sqlSourceDsn[0]       = NULLCHAR;
    outputString[0]       = NULLCHAR;

    /*****************************************************************
    * Get name of package to free                                    *
    *****************************************************************/
    getOptions( bindOptions,1024,"BINDOPTS" );

  } /* end of getDestroyData */


void getRebindData( void )            /* Rebind an SQL Procedure    */
  /*******************************************************************
  * Gets the name of the package to be rebound by DSNTPSMP during a  *
  * REBIND operation.                                                *
  *******************************************************************/
  {
    /*****************************************************************
    * Set program prep and runtime options to NULLCHAR               *
    *****************************************************************/
    sqlSource.length      = 0;                                /*@05*/
    sqlSource.data[0]     = NULLCHAR;                         /*@05*/
    precompOptions[0]     = NULLCHAR;
    compileOptions[0]     = NULLCHAR;
    prelinkOptions[0]     = NULLCHAR;
    linkOptions[0]        = NULLCHAR;
    alterStmt[0]          = NULLCHAR;                         /*@02*/
    sqlSourceDsn[0]       = NULLCHAR;
    outputString[0]       = NULLCHAR;

    /*****************************************************************
    * Get parameters to pass for rebind                              *
    *****************************************************************/
    getOptions( bindOptions,1024,"BINDOPTS" );

  } /* end of getRebindData */


void getLevelData( void )             /* QueryLevel of DSNTPSMP     */
  /*******************************************************************
  * Prepare for a DSNTPSMP QUERYLEVEL operation.                     *
  *******************************************************************/
  {
    /*****************************************************************
    * Set program prep and runtime options to NULLCHAR               *
    *****************************************************************/
    sqlSource.length      = 0;                                /*@05*/
    sqlSource.data[0]     = NULLCHAR;                         /*@05*/
    precompOptions[0]     = NULLCHAR;
    compileOptions[0]     = NULLCHAR;
    prelinkOptions[0]     = NULLCHAR;
    linkOptions[0]        = NULLCHAR;
    alterStmt[0]          = NULLCHAR;                         /*@02*/
    sqlSourceDsn[0]       = NULLCHAR;
    outputString[0]       = NULLCHAR;

```

```
  } /* end of getLevelData */


void getOptions                       /* Read processing options    */
  ( char *options,                    /* -out: list of options read */
    int maxBytes,                     /* - in: max size of list     */
    char *optionsDDname               /* - in: nameof DD to read    */
  )
  /******************************************************************
  * Reads up to maxBytes bytes of data from optionsDDname into the   *
  * options buffer.                                                  *
  ******************************************************************/
  { FILE           *optionsFile;      /* Ptr to specified options DD*/
    char           optionsDD[12];     /* DD handle                  */
    char           optionsRec[80];    /* Options file input record  */
    short int      recordLength  = 0; /* Length of record           */
    unsigned short moreRecords = Yes; /* EOF indicator              */

    sprintf( optionsDD,
             "DD:%s\0",
             optionsDDname );

    errno = 0;                        /* clear LE errno             */
    optionsFile = fopen( optionsDD,
                        "rb,lrecl=80,type=record" );
    if( optionsFile == NULL )
      issueDataSetOpeningError( optionsDD,errno );

    while( moreRecords == Yes  &&  rc < RETSEV )
      { recordLength
          = fread( optionsRec,        /* Read into options rec area */
                   1,                 /* ..1 record                 */
                   80,                /* ..of 80 bytes              */
                   optionsFile );     /* ..from current options file*/

        if( ferror(optionsFile) )     /* Handle IO errors           */
           issueDataSetReadingError( optionsDD,errno );

        else if( feof(optionsFile) )  /* Handle EOF                 */
           moreRecords = No;
                                      /* Discard bytes 73-80 and    */
        else                          /* strip off trailing blanks  */
          { strncat( options,optionsRec,72 );
            trimTrailingBlanks( options );
            /* Remove +/- continuation chars from BIND input     @03*/
            if( memcmp( optionsDDname,"BINDOPTS",8 ) == 0 )    /*@03*/
              stripContinuationCharacter( options );           /*@03*/
          }
                                      /* Don't overfill return area */
        if( rc < RETSEV  &&  strlen(options) > maxBytes )
          issueInvalidParmLengthError( optionsDD,0,maxBytes );
      }

    if( rc < RETSEV )
      if( fclose( optionsFile ) != 0 )
        issueDataSetClosingError( optionsDD,errno );

  } /* end of getOptions */


void getSqlSource( void )             /* Read SQL Procedure Source  */
  /******************************************************************
  * Reads up to 2M bytes of SQL Procedure source code from the       *
  * SQLIN DD.                                                        *
  ******************************************************************/
  { char           sourceRec[80];     /* Source file input record   */
    short int      recordLength  = 0; /* Length of record           */
    unsigned short moreRecords = Yes; /* EOF indicator              */

    /****************************************************************
    * Open the data set having the source for the SQL Procedure      *
    ****************************************************************/
    errno = 0;                        /* clear LE errno             */
    sqlInFile = fopen( "DD:SQLIN",
                       "rb,lrecl=80,type=record" );
    if( sqlInFile == NULL )
       issueDataSetOpeningError( "DD:SQLIN",errno );

    while( moreRecords == Yes  &&  rc < RETSEV )
      { recordLength
          = fread( sourceRec,         /* Read into source rec area  */
                   1,                 /* ..1 record                 */
```

```
                   80,                /* ..of 80 bytes              */
                   sqlInFile );       /* ..from SQL Proc source file*/

        if( ferror(sqlInFile) )       /* Handle IO errors           */
           issueDataSetReadingError( "DD:SQLIN",errno );

        else if( feof(sqlInFile) )    /* Handle EOF                 */
           moreRecords = No;
                                      /* Discard bytes 73-80, strip */
        else                          /* trailing blanks,add NL char*/
          { sourceRec[72] = NULLCHAR;
            trimTrailingBlanks( sourceRec );
            strncat( sourceRec,"\x25",1 );
            strcat( sqlSource.data, sourceRec );
            sqlSource.length = strlen(sqlSource.data);
          }
      /* Throw exception if not enough room for next record ...     */
      if( moreRecords == Yes && sqlSource.length >((2*1048576)-72) )
        issueInvalidParmLengthError( "DD:SQLIN",0,((2*1048576)-72) );
      }

    if( rc < RETSEV )
      if( fclose( sqlInFile ) != 0 )
        issueDataSetClosingError( "DD:SQLIN",errno );

  } /* end of getSQLsource */


void connectToLocation( void )        /* Connect to DB2 location    */
  /******************************************************************
  * Connects to the DB2 location specified in call parm number 4     *
  ******************************************************************/
  { EXEC SQL
      CONNECT TO :locationName;

    if( SQLCODE != 0 )
      { issueSqlError( "Connect to location failed" );
      }
  } /* end of connectToLocation */


void setAuthID( void )                /* Set the current DB2 auth id*/
  /******************************************************************
  * Changes the current authorization id to the one specified in     *
  * call parm number 3                                               *
  ******************************************************************/
  { EXEC SQL
      SET CURRENT SQLID = :authID;

    if( SQLCODE != 0 )
      { issueSqlError( "Set current SQLID failed" );
      }
  } /* end of setAuthID */


void callDSNTPSMP( void )             /* Run SQL Procedure Processor*/
  /******************************************************************
  * Calls the DSNTPSMP (DB2 SQL Procedures Processor)                *
  ******************************************************************/
  { listDSNTPSMPcallParms();

    EXEC SQL
      CALL SYSPROC.DSNTPSMP( :action,
                             :routineName,
                             :sqlSource,
                             :bindOptions,
                             :compileOptions,
                             :precompOptions,
                             :prelinkOptions,
                             :linkOptions,
                             :alterStmt,                       /*@02*/
                             :sqlSourceDsn,
                             :authID,                          /*@05*/
                             :DSNTPSMP_pname,                  /*@05*/
                             :outputString );

    /****************************************************************
    * Analyze status codes from DSNTPSMP                             *
    ****************************************************************/
    printf( "* DSNTPSMP has completed with return code %s\n",
               outputString );
    if( SQLCODE != 0  &&  SQLCODE != 466 )                    /*+@04*/
```

```
      { issueSqlError( "Call to DSNTPSMP failed" );
      }
    else if( levelquery != 'Y' )
      { DSNTPSMP_rc = atoi( outputString );
        if( SQLCODE == 466 )
          resultSetReturned = Yes;
        else /* SQLCODE == 0 */
          resultSetReturned = No;
      }                                                    /*-@04*/
    else /* levelquery == 'Y' */
      { DSNTPSMP_rc=0;                        /* not applicable */
        if( SQLCODE == 466 )
          resultSetReturned = Yes;
        else /* SQLCODE == 0 */
          resultSetReturned = No;
        /* Check that level returned matches to the TENTHS digit.   */
        if( memcmp( outputString,INTERFACE,3 ) != 0 )
          issueInvalidLevelError( outputString );
      }

  } /* end of callDSNTPSMP */


void listDSNTPSMPcallParms( void )    /* List parms sent to DSNTPSMP*/
  /*******************************************************************
  * Displays the arguments of parameters being passed to DSNTPSMP    *
  *******************************************************************/
  { printf( "****************************************"
            "***************************************\n" );
    printf( "* DSN8ED4 is now invoking the DB2 SQL Procedures "
            "Processor (SYSPROC.DSNTPSMP)\n" );
    printf( "*\n" );
    printf( "* Location name: %s\n", locationName );
    printf( "*\n" );
    printf( "* Action specified: %s\n", action );
    printf( "*\n" );
    printf( "* SQL Procedure name: %s\n", routineName );
    printf( "*\n" );
    printf( "* DB2 Precompiler Options:\n* %s\n", precompOptions );
    printf( "*\n" );
    printf( "* Compiler Options:\n* %s\n", compileOptions );
    printf( "*\n" );
    printf( "* Prelink Editor Options:\n* %s\n", prelinkOptions );
    printf( "*\n" );
    printf( "* Link Editor Options:\n* %s\n", linkOptions );
    printf( "*\n" );
    printf( "* DB2 Bind Options:\n* %s\n", bindOptions );
    printf( "*\n" );
    if( strlen(alterStmt) > 0 )                            /*@02*/
      {                                                    /*@02*/
        printf( "* ALTER statement:\n* %s\n", alterStmt );     /*@02*/
        printf( "*\n" );
      }                                                    /*@02*/

  } /* end of listDSNTPSMPcallParms */


void processDSNTPSMPresultSet( void ) /* Handle DSNTPSMP result sets*/
  /*******************************************************************
  * Outputs data from the result set returned by DSNTPSMP           *
  *******************************************************************/
  {
    /****************************************************************
    * Associate a locator with the result set from DSNTPSMP         *
    ****************************************************************/
    associateResultSetLocator();

    /****************************************************************
    * Allocate a cursor for the result set                          *
    ****************************************************************/
    if( rc < RETSEV )
      allocateResultSetCursor();

    /****************************************************************
    * Output reports returned in the result set                     *
    ****************************************************************/
    if( rc < RETSEV )
      writeDSNTPSMPreports();

  } /* end of processDSNTPSMPresultSet */

```

```
void associateResultSetLocator(void)  /* Assoc DSNTPSMP RS locator  */
  /*****************************************************************
  * Associates the result set from DSNTPSMP with a result set locator*
  *****************************************************************/
  { EXEC SQL
      ASSOCIATE
        LOCATORS( :DSNTPSMP_rs_loc1 )
      WITH PROCEDURE SYSPROC.DSNTPSMP;

    if( SQLCODE != 0 )
      { issueSqlError( "Associate locator call failed" );
      }

  } /* end of associateResultSetLocator */


void allocateResultSetCursor( void )  /* Alloc DSNTPSMP RS cursor   */
  /*****************************************************************
  * Allocates a cursor to the locator for the DSNTPSMP result set    *
  *****************************************************************/
  { EXEC SQL
      ALLOCATE DSNTPSMP_RS_CSR1
        CURSOR FOR RESULT SET :DSNTPSMP_rs_loc1;

    if( SQLCODE != 0 )
      { issueSqlError( "Allocate result set cursor "
                              "call failed" );
      }

  } /* end of allocateResultSetCursor */


void writeDSNTPSMPreports( void )     /* Print DSNTPSMP report      */
  /*****************************************************************
  * Outputs the reports returned in the result set from DSNTPSMP     *
  ******************************************************************
  * The result set returned by DSNTPSMP contains one or more reports.*
  *                                                                  *
  * Within the result set, reports are distinguished from one anoth- *
  * er by the STEP and FILE columns:                                 *
  * - STEP refers to the phase (e.g. precompile, compile, bind, etc.)*
  *   of DSNTPSMP that generated the report.                         *
  * - FILE distinguishes reports that are generated by the same STEP.*
  *                                                                  *
  * Report line data are stored in the LINE column, and arranged ac- *
  * cording to the sequence number in the SEQN column.               *
  *                                                                  *
  * In summary, STEPs contain FILEs, FILEs contain LINEs, and LINEs  *
  * are ordered according to SEQN (sequence).                        *
  *****************************************************************/
  { short int reportNumber = 1;       /* Sequence number of report  */
    char      prevStepName[17];       /* Track step name changes    */
    char      prevFileName[9];        /* Track file name changes    */
    short int recordLength = 0;       /* Length of record           */

    /***************************************************************
    * Get the first entry in the result set                        *
    ***************************************************************/
    fetchFromResultSetCursor();

    /***************************************************************
    * Allocate an outout DD for the first report                   *
    ***************************************************************/
    if( rc < RETSEV )
      openReportDataSet( reportNumber );

    /***************************************************************
    * Save step and file, to monitor for when they change          *
    ***************************************************************/
    if( rc < RETSEV )
      { strncpy( prevStepName,stepName,17 );
        strncpy( prevFileName,fileName,9 );
      }

    /***************************************************************
    * Process all rows in the result set                           *
    ***************************************************************/
    while( SQLCODE == 0  &&  rc < RETSEV )
      { if( ( strcmp( prevStepName,stepName ) != 0
            || strcmp( prevFileName,fileName ) != 0 )
          && reportNumber < 6 )                              /*@06*/
          /*********************************************************
```

```
          * If the step or file changes, allocate next report DD     *
          * up to and including report no. 6                      @06*
          **********************************************************/
          { closeReportDataSet();
            if( rc < RETSEV )
              openReportDataSet( ++reportNumber );
            if( rc < RETSEV )
              { strncpy( prevStepName,stepName,17 );
                strncpy( prevFileName,fileName,9 );
              }
          }
        /*************************************************************
        * Write the current report line to the current report DD     *
        ************************************************************/
        if( rc < RETSEV )
          { recordLength
              = fwrite( reportLine,   /* write from reportLine      */
                        1,            /* ..a record                 */
                        sizeof( reportLine ),
                        reportDD );   /* ..into the report data set */
          }
        if( rc < RETSEV )
          { fetchFromResultSetCursor();
          }
      }

    if( rc < RETSEV )
      { closeReportDataSet();
      }

  } /* end of writeDSNTPSMPreports */


void fetchFromResultSetCursor( void ) /* Read DSNTSPMP RS cursor    */
  /*******************************************************************
  * Reads the cursor for the DSNTPSMP result set                     *
  ******************************************************************/
  { memset( reportLine,' ',256 );

    EXEC SQL
      FETCH DSNTPSMP_RS_CSR1
      INTO  :stepName,
            :fileName,
            :reportLineNumber,
            :reportLine;

    if( SQLCODE != 0  &&  SQLCODE != 100  &&  rc < RETSEV )
      { issueSqlError( "*** Fetch from "
                                "result set cursor failed" );
      }
  } /* end of fetchFromResultSetCursor */


void openReportDataSet                /* Alloc DD for a report      */
  ( short int reportNumber            /* - in: Seqeunce number      */
  )
  /*******************************************************************
  * Opens the DD REPORTnn, where "nn" is the report number passed in *
  * and associates it with the file handler reportDD.                *
  ******************************************************************/
  { char          reportDDdcb[36];    /* for generated DCB          */

    if( reportNumber < 1 || reportNumber > 99 )
       issueInvalidDDnumError( reportNumber );

    else
      { sprintf( reportDDName,        /* Generate DD name REPORTnn  */
                 "DD:REPORT%2.2i\0",  /* ..where nn is the sequence */
                 reportNumber );      /* ..number of the report     */

        if( reportLine[0] == '1' )    /* Does this look like FBA?   */
          sprintf( reportDDdcb,       /* Yes: Specify               */
                   "wb,recfm=FBA,"    /* ..record output, recfm=fba */
                   "lrecl=256" );     /* ..and lrecl 255            */
        else
          sprintf( reportDDdcb,       /* No: Specify                */
                   "wb,recfm=FB,"     /* ..record output, recfm=fb  */
                   "lrecl=256" );     /* ..and lrecl 255            */

        errno = 0;                    /* clear LE errno             */
        reportDD = fopen( reportDDName,reportDDdcb );

```

```
        if( reportDD == NULL )        /* If unable to open data set */
          issueDataSetOpeningError( reportDDName,errno );
      }
  } /* end of openReportDataSet */


void closeReportDataSet( void )       /* Dealloc DD for a report    */
  /*******************************************************************
  * Closes the DD associated with the file handler reportDD.        *
  *******************************************************************/
  { if( fclose(reportDD) != 0 )
      issueDataSetClosingError( reportDDName,errno );
  } /* end of closeReportDataSet */


void trimTrailingBlanks               /* Strip off trailing blanks  */
  ( char            *string           /* - in: string to be trimmed */
  )
  /*******************************************************************
  * Strips trailing blanks from a string                            *
  *******************************************************************/
  { int             i;
    for( i = strlen(string) - 1; string[i] == ' '; i-- );
    string[++i] = '\0';
  } /* end of trimTrailingBlanks */

                                                         /*begin @03*/
void stripContinuationCharacter       /* Strip off trailing - or +  */
  ( char            *string           /* - in: string to be trimmed */
  )
  /*******************************************************************
  * Strips trailing '+' or '-' from a blank-trimmed string          *
  *******************************************************************/
  { int             i;
    i = strlen(string) - 1;
    if( string[i] == '+' || string[i] == '-' )
      string[i] = '\0';
    trimTrailingBlanks( string );
  } /* end of trimstripContinuationCharacter */
                                                           /*end @03*/

                                                         /*begin @04*/
void processSqlCommit( void )         /* Commit SQL changes         */
  /*******************************************************************
  * Commits the current unit of SQL work                            *
  *******************************************************************/
  { EXEC SQL
      COMMIT;

    if( SQLCODE != 0 )
      { issueSqlError( "*** Commit failed " );
      }
    else
      { printf( "* SQL changes have been committed\n" );
      }

  } /* end of processSqlCommit */


void processSqlRollback( void )       /* Rollback SQL changes       */
  /*******************************************************************
  * Rolls back the current unit of SQL work                         *
  *******************************************************************/
  { EXEC SQL
      ROLLBACK;

    if( SQLCODE != 0 )
      { issueSqlError( "*** Rollback failed " );
      }
    else
      { printf( "* SQL changes have been rolled back\n" );
      }

  } /* end of processSqlRollback */
                                                           /*end @04*/

void issueDataSetClosingError         /* Handler for ds close error */
  ( char            *DDname,          /* - in: name of errant DD    */
    int             LEerrno           /* - in: LE diagnostic errno  */
  )
  /*******************************************************************
  * Called when a TSO data set cannot be closed                     *
```

```
  *****************************************************************/
  { printf( "ERROR: Unable to close %s\n", DDname );
    printf( "%s \n",strerror(LEerrno) );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueDataSetClosingError */


void issueDataSetOpeningError         /* Handler for ds open error  */
  ( char            *DDname,          /* - in: name of errant DD    */
    int             LEerrno           /* - in: LE diagnostic errno  */
  )
  /*****************************************************************
  * Called when a TSO data set cannot be opened                     *
  *****************************************************************/
  { printf( "ERROR: Unable to open %s\n", DDname );
    printf( "%s \n",strerror(LEerrno) );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueDataSetOpeningError */


void issueDataSetReadingError         /* Handler for ds read error  */
  ( char            *DDname,          /* - in: name of errant DD    */
    int             LEerrno           /* - in: LE diagnostic errno  */
  )
  /*****************************************************************
  * Called when a TSO data set cannot be read                       *
  *****************************************************************/
  { printf( "ERROR: Unable to read %s\n", DDname );
    printf( "%s \n",strerror(LEerrno) );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueDataSetReadingError */


void issueInvalidCallParmCountError   /* Handler for parm count err */
  ( int argc                          /* - in: no. parms received   */
  )
  /*****************************************************************
  * Called when this program is invoked with an inappropriate number *
  * of call parms.                                                  *
  *****************************************************************/
  { printf( "ERROR: DSN8ED4 was invoked with %i parameters\n",--argc );
    printf( "       - The first three parms (action, routine "
                    "name, and authid) are required\n" );
    printf( "       - The fourth parm (location name) "
                    "is optional\n" );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidCallParmCountError */


void issueInvalidDDnumError           /* Handler for unknown DD seq */
  ( short          invalidDDnum       /* - in: invalid DD sequ. no. */
  )
  /*****************************************************************
  * Called when the sequence number for a report DD (REPORTnn, where *
  * "nn" is the sequence number" is less than 1 or greater 99.      *
  *****************************************************************/
  { printf( "ERROR: Invalid sequence "/* Issue error messages       */
            "number <%i> specified "  /* ..for DD REPORTnn          */
            "for REPORTnn DD\n",      /* ..where nn is the sequence */
            invalidDDnum );           /* ..number of the result set */
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidDDnumError */


void issueInvalidActionError          /* Handler for unknown action */
  ( char *action                      /* - in: action specified     */
  )
  /*****************************************************************
  * Called when an unexpected argument is specified for the DB2 SQL  *
  * Procedures Processor action                                     *
  *****************************************************************/
  { printf( "ERROR: The argument specified for the action "
                   "parameter is invalid\n",action );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidActionError */
```

```

void issueInvalidParmLengthError      /* Handler for parm len error */
  ( char *parmName,                   /* - in: identify of parm     */
    int minLength,                    /* - in: min valid length     */
    int maxLength                     /* - in: max valid length     */
  )
  /*****************************************************************
  * Called when the length of an argument specified for a DSNTPSMP   *
  * parameter (parmName) does not fall within the valid bounds for   *
  * size (minLength and maxLength) for that parameter                *
  *****************************************************************/
  { printf( "ERROR: The length of the argument specified for the %s "
                   "parameter\n",parmName );
    printf( "       does not fall within the required bounds of %i "
                   "and %i\n",minLength,maxLength );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidParmLengthError */


void issueInvalidLevelError           /* Handler for wrong DSNTPSMP */
  ( char *level                       /* - in: level encountered    */
  )
  /*****************************************************************
  * Called when a DSNTPSMP QUERYLEVEL request returns a level not    *
  * handled by this sample client.                                   *
  *****************************************************************/
  { printf( "ERROR: The DSNTPSMP interface level %s is not "
                   "supported by this client\n",level );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidLevelError */


#pragma linkage(dsntiar, OS)
void issueSqlError                    /* Handler for SQL error      */
  ( char *locMsg                      /* - in: Call location        */
  )
  /*****************************************************************
  * Called when an unexpected SQLCODE is returned from a DB2 call    *
  *****************************************************************/
  { struct      error_struct {        /* DSNTIAR message structure  */
      short int   error_len;
      char        error_text[10][80];
      }         error_message = {10 * 80};

    extern short int dsntiar( struct      sqlca         *sqlca,
                              struct      error_struct  *msg,
                              int                       *len );

    short int   DSNTIARrc;            /* DSNTIAR Return code        */
    int         j;                    /* Loop control               */
    static int  lrecl = 80;           /* Width of message lines     */

    /***************************************************************
    * print the locator message                                      *
    ***************************************************************/
    printf( "ERROR: %-80s\n", locMsg );
    printf( "-----> Processing halted\n" );

    /***************************************************************
    * format and print the SQL message                               *
    ***************************************************************/
    DSNTIARrc = dsntiar( &sqlca, &error_message, &lrecl );
    if( DSNTIARrc == 0 )
      for( j = 0; j <= 10; j++ )
        printf( " %.80s\n", error_message.error_text[j] );
    else
      {
        printf( " *** ERROR: DSNTIAR could not format the message\n" );
        printf( " ***        SQLCODE is %d\n",SQLCODE );
        printf( " ***        SQLERRM is \n" );
        for( j=0; j<sqlca.sqlerrml; j++ )
          printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
      }

    /***************************************************************
    * set severe error code                                          *
    ***************************************************************/
    rc = RETSEV;
```

```
                                                                     11890000
   } /* end of issueSqlError */                                      11900000
```

Related reference

“Sample programs to help you prepare and run external SQL procedures” on page 307
Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

DSN8WLMP

This JCL can be customized to establish the WLM startup PROC needed to run DSNTPSMP, the Db2 SQL Procedures Processor, and to run ADMIN_UPDATE_SYSPARM, the Db2 stored procedure that changes subsystem parameters.

```
//*********************************************************************
//*  Name = DSN8WLMP
//*
//*  Descriptive Name =
//*    DB2 Sample WLM startup PROC for DSNTSPMP, the DB2 SQL Procedures
//*    Processor, and for ADMIN_UPDATE_SYSPARM, the DB2 stored
//*    procedure that changes subsystem parameters.
//*
//*
//*    Licensed Materials - Property of IBM
//*    5635-DB2
//*    (C) COPYRIGHT 1982, 2006 IBM Corp.  All Rights Reserved.
//*
//*    STATUS = Version 11
//*
//*  Function =
//*    This JCL can be customized to establish the WLM startup PROC
//*    needed to run DSNTPSMP, the DB2 SQL Procedures Processor,
//*    and to run ADMIN_UPDATE_SYSPARM, the DB2 stored procedure
//*    that changes subsystem parameters.
//*
//*    Before you can use this procedure, you need to have defined a
//*    WLM Application Environment for running DSNTPSMP and
//*    ADMIN_UPDATE_SYSPARM.
//*
//*    *** *** *** *** *** *** IMPORTANT *** *** *** *** *** *** ***
//*    For DSNTPSMP and ADMIN_UPDATE_SYSPARM, NUMTCB=1 is required.
//*    Specify no other value. This assures concurrent executions
//*    of DSNTPSMP and ADMIN_UPDATE_SYSPARM will run in their
//*    own address space, which is needed for proper dataset
//*    operation from within a REXX/TSO DB2 stored procedure.
//*
//*    (1) Customize this proc for use on your system by locating and
//*        changing all occurrences of the following strings as
//*        indicated:
//*        (A) '!WLMENV!' to the name of the WLM Application Environment
//*            you have chosen for running DSNTPSMP and
//*            ADMIN_UPDATE_SYSPARM
//*        (B) '!DSN8WLMP!' to the name of the WLM Procedure associated
//*            with that environment
//*        (C) '!DSN!' to the name of your DB2 subsystem
//*        (D) 'CBC!!' to the prefix of your target library for
//*            IBM C/C++ for z/OS
//*        (E) 'CEE!!' to the prefix of your target library for
//*            IBM Language Environment for z/OS
//*        (F) 'DSN!!0' to the prefix of your target library for
//*            DB2 for z/OS
//*    (2) Copy the customized proc to your MVS proclib, to the member
//*        you specified as the WLM procedure name for the WLM
//*        application environment you have chosen for running DSNTPSMP
//*        and ADMIN_UPDATE_SYSPARM
//*        Note: This should be the same value as you specified in
//*              step 1B, above.
//*
//* CHANGE LOG:
//*   09/20/2012 Add ZPMDFLTS for ADMIN_UPDATE_SYSPARM DK1557/PM71114
//*
//*********************************************************************
//!DSN8WLMP! PROC DB2SSN=!DSN!,NUMTCB=1,APPLENV=!WLMENV!
//*
//NUMTCB@1 SET NUMTCB=                         <== Null NUMTCB symbol
//*
//DSNTPSMP EXEC PGM=DSNX9WLM,TIME=1440,
//             PARM='&DB2SSN,1,&APPLENV',      <== Use 1, not NUMTCB
```

```
//             REGION=0M,DYNAMNBR=5            <== Allow for Dyn Allocs
//* Include SDSNEXIT to use Secondary Authids (DSN3@ATH DSN3@SGN exits)
//STEPLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//         DD  DISP=SHR,DSN=CBC!!.SCCNCMP      <== C Compiler
//         DD  DISP=SHR,DSN=CEE!!.SCEERUN      <== LE runtime
//SYSEXEC  DD  DISP=SHR,                       <== Location of DSNTPSMP
//             DSN=DSN!!0.SDSNCLST                 and DSNADMUZ
//SYSTSPRT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSABEND DD  DUMMY
//DSNTRACE DD  SYSOUT=*
//*
//**** Data sets required by the SQL Procedures Processor
//SQLDBRM  DD  DISP=SHR,                       <== DBRM Library
//             DSN=DSN!!0.DBRMLIB.DATA
//SQLCSRC  DD  DISP=SHR,                       <== Generated C Source
//             DSN=DSN!!0.SRCLIB.DATA
//SQLLMOD  DD  DISP=SHR,                       <== Application Loadlib
//             DSN=DSN!!0.RUNLIB.LOAD
//SQLLIBC  DD  DISP=SHR,                       <== C header files
//             DSN=CEE!!.SCEEH.H
//         DD  DISP=SHR,
//             DSN=CEE!!.SCEEH.SYS.H
//         DD  DISP=SHR,                       <== Debug header file
//             DSN=DSN!!0.SDSNC.H
//SQLLIBL  DD  DISP=SHR,                       <== Linkedit includes
//             DSN=CEE!!.SCEELKED
//         DD  DISP=SHR,
//             DSN=DSN!!0.SDSNLOAD
//SYSMSGS  DD  DISP=SHR,                       <== Prelinker msg file
//             DSN=CEE!!.SCEEMSGP(EDCPMSGE)
//*
//**** DSNTPSMP Configuration File - CFGTPSMP (optional)
//*             A site provided sequential dataset or member, used to
//*             define customized operation of DSNTPSMP in this APPLENV.
//*CFGTPSMP DD  DISP=SHR,DSN=
//*
//**** Workfiles required by the SQL Procedures Processor
//SQLSRC   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLPRINT DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLTERM  DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLOUT   DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLCPRT  DD  UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLUT1   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLUT2   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLCIN   DD  UNIT=SYSALLDA,SPACE=(32000,(20,20))
//SQLLIN   DD  UNIT=SYSALLDA,SPACE=(3200,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SQLDUMMY DD  DUMMY
//SYSMOD   DD  UNIT=SYSALLDA,SPACE=(23440,(20,20)),   <= PRELINKER
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//*
//**** Data sets required by ADMIN_UPDATE_SYSPARM
//ZPMDFLTS DD  DISP=SHR,                       <== Defaults file
//             DSN=DSN!!0.NEW.SDSNSAMP(DSNADMZW)
//*
```

Related reference

“Sample programs to help you prepare and run external SQL procedures” on page 307
Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

DSN8ED5
Demonstrates how to call the sample SQL procedure DSN8.

```
/********************************************************************* 00010000
* Module name = DSN8ED5 (DB2 sample program)                         * 00020000
*                                                                    * 00030000
* DESCRIPTIVE NAME = Client for sample SQL Procedure DSN8.DSN8ES2    * 00040000
```

```
*                                                                     * 00050000
*                                                                     * 00060000
*    LICENSED MATERIALS - PROPERTY OF IBM                             * 00070000
*    5675-DB2                                                         * 00080000
*    (C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED.          * 00100000
*                                                                     * 00130000
*    STATUS = VERSION 7                                               * 00140000
*                                                                     * 00170000
* Function: Demonstrates how to call the sample SQL procedure        * 00230000
*           DSN8.DSN8ES2 using static SQL.                            * 00240000
*                                                                     * 00250000
* Notes:                                                              * 00260000
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       * 00270000
*                                                                     * 00280000
*   Restrictions:                                                     * 00290000
*                                                                     * 00300000
* Module type: C program                                              * 00310000
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                    * 00320000
* Module size: See linkedit output                                    * 00330000
*  Attributes: Re-entrant and re-usable                               * 00340000
*                                                                     * 00350000
* Entry Point: DSN8ED5                                                * 00360000
*     Purpose: See Function                                           * 00370000
*     Linkage: Standard MVS program invocation, one parameter.       * 00380000
*                                                                     * 00390000
*                                                                     * 00400000
*  Parameters: DSN8ED5 uses the C "main" argument convention of      * 00410000
*              argv (argument vector) and argc (argument count).     * 00420000
*                                                                     * 00430000
*              - ARGV[0]: (input) pointer to a char[9],               * 00440000
*                         null-terminated string having the name of  * 00450000
*                         this program (DSN8ED5)                      * 00460000
*              - ARGV[1]: (input) pointer to a char[10],              * 00470000
*                         null-terminated string that contains the   * 00480000
*                         amount of the base bonus for sample        * 00490000
*                         managers.  The format is: nnnnnn.nn        * 00500000
*              - ARGV[2]: (input) pointer to a char[17],              * 00510000
*                         null-terminated string having the name of  * 00520000
*                         the server where DSN8.DSN8ES2 resides.     * 00530000
*                         This is an optional parameter; the local   * 00540000
*                         server is used if no argument is provided. * 00550000
*                                                                     * 00560000
* Normal Exit: Return Code: 0000                                      * 00570000
*              - Message: none                                        * 00580000
*                                                                     * 00590000
*  Error Exit: Return Code: 0008                                      * 00600000
*              - Message: DSN8ED5 failed: Invalid parameter count    * 00610000
*              - Message: DSN8ED5 failed: Argument to parameter 1    * 00620000
*                                         exceeds 9 bytes             * 00630000
*              - Message: DSN8ED5 failed: No result from DSN8.DSN8ES2* 00640000
*              - Message: <formatted SQL text from DSNTIAR>          * 00650000
*                                                                     * 00660000
*                                                                     * 00670000
*    External References:                                             * 00680000
*             - Routines/Services: DSNTIAR: DB2 msg text formatter   * 00690000
*             - Data areas       : None                               * 00700000
*             - Control blocks   : None                               * 00710000
*                                                                     * 00720000
*  Pseudocode:                                                        * 00730000
*   DSN8ED5:                                                          * 00740000
*   - Verify that 2 or 3 input parameters (program name, base bonus  * 00750000
*     amount and, optionally, remote location name) were passed.     * 00760000
*     - if not, issue diagnostic message and end with code 0008      * 00770000
*   - Connect to the remote location, if one was specified           * 00780000
*   - Call sample SQL Procedure DSN8.DSN8ES2, passing the base bonus * 00790000
*     amount as the argument of the first (input) parameter.         * 00800000
*     - if unsuccessful, call sql_error to issue a diagnostic mes-   * 00810000
*       sage, then end with code 0008.                                * 00820000
*   - Report the value returned by DSN8.DSN8ES2 in its second        * 00830000
*     (output) parameter.                                             * 00840000
*   End DSN8ED5                                                       * 00850000
*                                                                     * 00860000
*   sql_error:                                                        * 00870000
*   - call DSNTIAR to format the unexpected SQLCODE.                 * 00880000
*   End sql_error                                                     * 00890000
*                                                                     * 00900000
*********************************************************************/ 00910000
/********************** C library definitions ***********************/ 00920000
#include <stdio.h>                                                      00930000
#include <stdlib.h>                                                     00940000
#include <string.h>                                                     00950000
#include <decimal.h>                                                    00960000
```

```
/****************************** Equates ******************************/
#define         NULLCHAR     '\0'     /* Null character             */

#define         OUTLEN        80      /* Length of output line      */
#define         DATA_DIM      10      /* Number of message lines    */

#define         NOT_OK         0      /* Run status indicator: Error*/
#define         OK             1      /* Run status indicator: Good */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
  char          locationName[17];     /* Server location name       */

  decimal(15,2) hvBonusBase     = 0;  /* base bonus for managers    */
  short int     niBonusBase     = 0;  /* Indic var for hvBonusBase  */

  decimal(15,2) hvBonuses       = 0;  /* tot bonuses rtnd by DSN8ES2*/
  short int     niBonuses       = 0;  /* Indic var for hvBonuses    */

  long int      hvSqlErrCd      = 0;  /* Err SQLCODE from DSN8ES2   */
  short int     niSqlErrCd      = 0;  /* Indic var for hvSqlErrCd   */

EXEC SQL END DECLARE SECTION;


/********************** DB2 Message Formatter **********************/
struct          error_struct          /* DSNTIAR message structure  */
{
  short int     error_len;
  char          error_text[DATA_DIM][OUTLEN];
}               error_message = {DATA_DIM * (OUTLEN)};

#pragma         linkage( dsntiar, OS )

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );


/********************* DSN8ED5 Global Variables ********************/
short int       status = OK;          /* DSN8ED5 run status         */

long int        completion_code = 0;  /* DSN8ED5 return code        */


/******************** DSN8ED5 Function Prototypes *******************/
int main( int argc, char *argv[] );
void sql_error( char locmsg[] );


int main( int argc, char *argv[] )
/*********************************************************************
* Get input parms, pass them to DSN8ES2, and process the results     *
*********************************************************************/
{
  printf( "**** DSN8ED5: Sample client for DB2 SQL Procedure Sample "
                        "(DSN8.DSN8ES2)\n\n" );
  printf( "*\n" );

  if( argc < 2  ||  argc > 3 )
    {
      printf( "DSN8ED5 failed: Invalid parameter count\n" );
      status = NOT_OK;
    }
  else if( strlen(argv[1]) > 9 )
    {
      printf( "DSN8ED5 failed: Bonus base exceeds 9 bytes.  "
                              "Use format: nnnnnn.nn\n" );
      status = NOT_OK;
    }
  else
    { /* Convert the input parameter from a string to a decimal */
      hvBonusBase = atof( argv[1] );
    }

  /*******************************************************************
```

```
  * Validate remote location name, if one is specified               *
  *******************************************************************/
  if( argc == 3  &&  status == OK )
    if( strlen( argv[2] ) < 1  ||  strlen( argv[2] ) > 16 )
      {
        printf( "DSN8ED5 failed: Length of location name must be "
                                "1 to 16 bytes\n" );
        status = NOT_OK;
      }
    else
      {
        strcpy( locationName,argv[2] );
        printf( "* Processing at location: %s\n",locationName );
        printf( "*\n" );
      }
  else
    locationName[0] = NULLCHAR;

  if( status == OK )
    {
      printf( "* Base bonus amount: %D(15,2)\n",hvBonusBase );
      printf( "*\n" );
    }

  /*******************************************************************
  * Connect to the remote location, if one was specified             *
  *******************************************************************/
  if( strlen(locationName) > 0  &&  status == OK )
    {
      EXEC SQL CONNECT TO :locationName;
      if( SQLCODE != 0 )
        sql_error( " *** Connect to remote server" );
    }

  /*******************************************************************
  * Process the call to DSN8.DSN8ES2                                 *
  *******************************************************************/
  if( status == OK )
    {
      EXEC SQL CALL DSN8.DSN8ES2( :hvBonusBase :niBonusBase,
                                  :hvBonuses   :niBonuses,
                                  :hvSqlErrCd  :niSqlErrCd );
      if( SQLCODE != 0 )
        sql_error( " *** Call DSN8.DSN8ES2" );
      else if( niSqlErrCd == 0 )
        {
          printf( "DSN8ED5 failed: Error SQLCODE from DSN8.DSN8ES2 "
                                  "is %i\n", hvSqlErrCd );
          status = NOT_OK;
        }
      else if( niBonuses != 0 )
        {
          printf( "DSN8ED5 failed: No result from DSN8.DSN8ES2\n" );
          status = NOT_OK;
        }
      else
        {
          printf( "* Total bonuses paid to management: $%D(15,2)\n",
                  hvBonuses );
        }
    }

  if( status != OK )
    completion_code = 8;

  return( completion_code );

} /* end main */


/*********************************************************************
**********************************************************************
** SQL error handler                                                **
**********************************************************************
*********************************************************************/
void sql_error( char locmsg[] )                               /*proc*/
{


  short int   rc;                     /* DSNTIAR Return code        */
  int         j,k;                    /* Loop control               */
  static int  lrecl = OUTLEN;         /* Width of message lines     */
```

```
   /****************************************************************
   * set status to prevent further processing                      *
   ****************************************************************/
   status = NOT_OK;

   /****************************************************************
   * print the locator message                                     *
   ****************************************************************/
   printf( " %.80s\n", locmsg );

   /****************************************************************
   * format and print the SQL message                              *
   ****************************************************************/
   rc = dsntiar( &sqlca, &error_message, &lrecl );
   if( rc == 0 )
     for( j=0; j<DATA_DIM; j++ )
       {
         for( k=0; k<OUTLEN; k++ )
           putchar(error_message.error_text[j][k] );
         putchar('\n');
       }
   else
     {
       printf( " *** ERROR: DSNTIAR could not format the message\n" );
       printf( " ***        SQLCODE is %d\n",SQLCODE );
       printf( " ***        SQLERRM is \n" );
       for( j=0; j<sqlca.sqlerrml; j++ )
         printf( "%c", sqlca.sqlerrmc[j] );
       printf( "\n" );
     }

 } /* end of sql_error */
```

Related reference

"Sample programs to help you prepare and run external SQL procedures" on page 307

Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

DSNTEJ67

This job demonstrates two important steps to follow when considering the conversion of an external SQL procedure to a native SQL procedure.

```
//**********************************************************************
//*  Name = DSNTEJ67
//*
//*  Descriptive Name =
//*    DB2 Sample Application
//*    Phase 6
//*    REXX and SQL PL
//*
//*    Licensed Materials - Property of IBM
//*    5615-DB2
//*    (C) COPYRIGHT 2010, 2013 IBM Corp.  All Rights Reserved.
//*
//*    STATUS = Version 11
//*
//*  Function = This job demonstrates two important steps to follow when
//*             considering the conversion of an external SQL procedure
//*             to a native SQL procedure. It all begins with a copy of
//*             the external SQL procedure source:
//*               1) Modify the SQL procedure options in the source
//*                  a) REMOVE options that relate only to external
//*                     SQL procedures
//*                  b) ADD native SQL PL options that relate to DB2
//*                     precompiler options
//*                  c) ADD native SQL PL options that relate to DB2
//*                     BIND PACKAGE options
//*               2) Review the SQL procedure source logic. Address
//*                  any identified syntax issues or published semantic
//*                  incompatibilities.
//*
//*  Pseudocode =
//*   This sample assists in this activity by performing the following:
//*
//*   PH067S00 Step
//*     Define the DB2 SSID to use for this job.
//*   PH067S01 Step
```

```
//*     Define Input. Identify the name of an external SQL SP with
//*                   source saved in DB2 (SYSIBM.SYSROUTINES_SRC).
//*   PH067S02 Step
//*     Define Output. Specify an output data set where the extracted
//*                    and modified SQL SP source is to be placed.
//*   PH067S03 Step
//*     Setup the sample REXX services to used for this job.
//*   PH067S04 Step
//*     Execute the DSNTEJ67 sample conversion process.
//*     - Validate the SP name              (using the NAMPARTS service)
//*     - Verify the output file is usable  (using the CHKANYFV service)
//*     - Deploy DSN8EN1, a sample native
//*       SQL SP helper for use later       (using the CRSQLPL  service)
//*     - Extract external SQL SP source    (using the SQLPLSRC service)
//*     - Save the source in the output file
//*       - for a RECFM V output file       (using the ANY2SQLV service)
//*       - for a RECFM F output file       (using the SQLV2F   service)
//*     - Validate and inspect the source   (using the CHKSQLPL service)
//*     - Produce a table of contents to
//*       describe the DDL syntax elements
//*       present in the SQL PL source      (using the SQLPLTOC service)
//*     - Dissect the external SQL SP source
//*       removing all the SP options
//*     - Get the replacement options for
//*       native SQL PL use by calling the
//*       helper SQL SP deployed earlier    (using the SQLCALL  service)
//*     - Reassemble the SQL SP source as a
//*       string and write it to a RECFM V
//*       temporary file (aka SQLV)         (using the STR2SQLV service)
//*     - Update the output file
//*       - for a RECFM V output            (using the ANY2SQLV service)
//*       - for a RECFM F output            (using the SQLV2F   service)
//*     - Write a special format temp file
//*       (aka s80) for the precompiler     (using the SQLV2F   service)
//*     - Obtain a HOST(SQLPL) Checkout
//*       precompiler listing of the SQL SP
//*       source for job log output.        (using the CHKSQLPL service)
//*     - Set the Job step RC.
//*
//*
//*  Dependencies =
//*    (1) Run sample job DSNTEJ65 prior to running this job.
//*        That job uses the DB2 SQL procedure processor DSNTPSMP to
//*        deploy the sample external SQL procedure DSN8.DSN8ES2, which
//*        is the external SQL Procedure this job processes by default.
//*
//*  Notes =
//*    Prior to running this job, customize it for your system:
//*    (1) Add a valid job card
//*    (2) Locate and change all occurrences of the following strings
//*        as indicated:
//*        (A) '!DSN!'      to the subsystem name of your DB2. This is
//*                         located in Step 0.
//*        (B) 'DSN!!0'     to the prefix of the target library for the
//*                         current DB2 release. This is located in the
//*                         JOBLIB, Step 3 and Step 4.
//*    (3) (Optional) Change either of the following to customize the
//*        input and output of this job for your particular purposes:
//*
//*        (A)  Change the name of the external SQL procedure to be
//*             processed by this job. This is defined in job Step 1.
//*             The name must include the schema qualifier. It must
//*             designate an operational external SQL SP which was
//*             deployed using the DB2 SQL procedure processor DSNTPSMP.
//*
//*        (B)  Change the data set where the extracted and modified
//*             SQL procdure source will be written. This is defined in
//*             job Step 2. The specification can be for an existing
//*             data set or data set member, qualified or not qualified.
//*             or represented by a DD descriptor (in the form of
//*             DD:ddname). Any existing data set or ddname allocation
//*             must be for a RECFM=F,FB,V,VB sequential data set or
//*             data set member. (If an unallocated ddname is provided
//*             a temporary sequential data set will be allocated.)
//*
//*  Change Activity = ( V11 base pm29226a )
//*********************************************************************
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//*
//*********************************************************************
//*  Step 0:  Store the default DB2 System SSID in a temporary data set
```

```
//*           for use by various steps and services that are run.
//*********************************************************************
//PH067S00 EXEC PGM=IEBGENER
//SYSUT1   DD *     Enter the desired DB2 SSID or Group attachment name
   !DSN!
//SYSUT2   DD DSN=&&PARM0,DISP=(NEW,PASS),SPACE=(TRK,1),
//            DCB=(LRECL=80,RECFM=FB,BLKSIZE=160)
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//*********************************************************************
//*  Step 1:  Store the desired external SQL SP name to processs.
//*           Specify a fully qualified SP name (2-parts, schema+name).
//*********************************************************************
//PH067S01 EXEC PGM=IEBGENER
//SYSUT1   DD *            For a long name, wrap the input at column 72
   DSN8.DSN8ES2
//*-+----1----+----2----+----3----+----4----+----5----+----6----+----7--
//SYSUT2   DD DSN=&&PARM1,DISP=(NEW,PASS),SPACE=(TRK,1),
//            DCB=(LRECL=72,RECFM=FB,BLKSIZE=576)
//SYSPRINT DD DUMMY
//SYSIN    DD *
     GENERATE MAXFLDS=1
       RECORD FIELD=(72)
//*********************************************************************
//*  Step 2:  Identify the desired data set name to store the extracted
//*           and modified SQL procedure source. Must be Recfm F or V,
//*           sequential or member, or a non-existing data set/member.
//*           The spcification can be a qualified name, a non-qualified
//*           name or a DD descriptor (in the form of DD:ddname).
//*********************************************************************
//PH067S02 EXEC PGM=IEBGENER
//SYSUT1   DD *
   DD:TEMPSRC
//SYSUT2   DD DSN=&&PARM2,DISP=(NEW,PASS),SPACE=(TRK,1),
//            DCB=(LRECL=72,RECFM=FB,BLKSIZE=576)
//SYSPRINT DD DUMMY
//SYSIN    DD *
     GENERATE MAXFLDS=1
       RECORD FIELD=(72)
//*********************************************************************
//*  Step 3:  Populate a temporary PDS with REXX services used locally
//*********************************************************************
//PH067S03 EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD DUMMY
//SYSUT2   DD DSN=&&REXXPDS,DISP=(NEW,PASS),
//            SPACE=(TRK,(5,5,2)),DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSIN    DD DSN=DSN!!0.SDSNMACS(DSN8ERL1),
//            DISP=SHR
//         DD DATA,DLM='@@'
./  ADD NAME=DSNTEJ67
/* ***** REXX ******** DSNTEJ67 command *************************** */
address TSO
PREPSSID '. V9 NFM'
if rc>=8 then do;
  Say 'DSNTEJ67 Unable to establish a connection to DB2';
  exit 8;
  end;

/* From DD:SPNAME read the stored procedure name to extract from DB2.
 * The name must be a schema qualified SP name (2-part name).
 * The SP must be for an external SQL procedure, with source in DB2.
 */
'EXECIO * DISKR SPNAME (OPEN FINIS STEM TEMP.';
spname='';
do i = 1 to TEMP.0;
 spname = spname || TEMP.i;
 end;
spname = "STRIP"(spname,'B');
/* Process the passed name to get the name parts,
 * plus the string and delimited forms.
 */
parse value "NAMPARTS"( spname ) with p# . namSpec
if p#<>2 then do;
  say 'DSNTEJ67 the passed SP name' spname,
              'was not schema qualified (2-parts)'
  exit 8;
  end;
parse var namSpec a b c d e . ':' +1 sNam +(a) sSch +(b) . +(c),
                                     qNam +(d) qSch +(e)
spname = qSch'.'qNam             /* 2-part fully qualified form now */

/* From DD:SOURCEDS read the data set specification for where the
```

```
 * extracted and modified SQL proc source should be written at JOB end.
 */
'EXECIO * DISKR SOURCEDS (OPEN FINIS STEM TEMP.';
sourceFile='';
do i = 1 to TEMP.0;
 sourceFile = sourceFile || TEMP.i;
 end;
sourceFile = "STRIP"(sourceFile,'B');

/* Find the status of the target data set for the source. It must be a
 * Sequential data set, F or V record format (or capable of same).
 */
parse value "CHKANYFV"(sourceFile) with oRecfm oLrecl oEmpty;
if oRecfm='' then do;
  say 'DSNTEJ67 Cannot use the designated data set' sourceFile,
               'for SQL PL source'
  exit 8;
  end;

prepConv:
/* From DD:HELPRSP1 deploy the native SQL SP DSN8.DSN8EN1 that will
 * provide native options for the external SQL proc to be migrated.
 */
Say 'Setting up the native SQL PL helper routine...'
'CRSQLPL' 'DD:HELPRSP1';
if RC>4 then do;
  say 'DSNTEJ67 cannot deploy the native SP used for migration.';
  exit 8;
  end;

allocList='';                       /* List of DDnames we allocate */

extractSQLproc:
/* Extract the SP source to a temporary SQLV file. Also get an S80
 * format edition to use with the precompiler for inspection purposes.
 */
'SQLPLSRC' spname 'DD:SQLVE' 'DD:S80E' 'ASIS';
if RC>4 then do;
  say 'DSNTEJ67 Cannot extract SQL procedure source';
  exit 8;
  end;
allocList='SQLVE,S80E';                          /* allocated FOR us */
/* Write extracted source ASIS now to the output data set.
 * We will rewrite it again later after reaching the point of editing.
 */
if oRecfm='F' then do;
     if oLrecl=80 then seq='SEQ'; else seq='';
     "SQLV2F" 'DD:SQLVE' sourceFile seq
     end;
else "ANY2SQLV" 'DD:SQLVE' sourceFile 'EXTEND'
if RC<>0 then do;
  say 'DSNTEJ67 Cannot write extracted source to data set' sourceFile
  exit 8;
  end;
else say 'Source for SP' spname 'written to data set' sourceFile

verifyExtSQL:
/* Verify the extracted source is valid external SQL procedure source
 * before going to far. Use the HOST(SQL) precompiler.
 */
/* Keep DD:LISTING active till then end. */
'ALLOCATE DDNAME(LISTING) NEW REUSE';
'CHKSQLPL' 'DD:S80E' 'DD:LISTING' '.' 'MAR(1,80) HOST(SQL)'
if RC>4 then do;
  msg='DSNTEJ67 Extracted external SQL procedure source has errors';
  call endWithListing msg, allocList;
  end;

chkoutSQLPL1:
/* Inspect the extracted external SQL procedure source without change
 * using the HOST(SQLPL) Checkout precompiler.
 * RC=8 errors are anticipated.
 */
allocList = allocList||',UT1';
'ALLOCATE DDNAME(UT1) NEW REUSE';
'CHKSQLPL' 'DD:S80E' 'DD:LISTING' 'DD:UT1' 'MAR(1,80)'
if RC>8 then do;
  msg='DSNTEJ67 Fatal error running HOST(SQLPL) precompiler',
               'with external SQL procedure source'
  call endWithListing msg, allocList;
  end;
/* Getting no UT1 content typically represents a native SQL PL syntax
```

```
 * issue. In this context, that could be caused by some unforeseen
 * difference between valid external SQL PL and native SQL PL syntax.
 */
parse value "CHKANYFV"('DD:UT1') with utR . utE;
if utE<>'1' then do;
  msg='DSNTEJ67 Syntax error in external SQL proc source',
               'when viewed as native SQL PL';
  call endWithListing msg, allocList;
  end;

editPrep:
/* Obtain an SQLPL TOC description, to use for editing the source. */
'SQLPLTOC' 'DD:S80E' 'DD:UT1' 'DD:TOC'
if RC<>0 then do;
  msg='DSNTEJ67 Unable to prepare for source editing (no TOC).'
  call endWithListing msg, allocList;
  end;
allocList = allocList||',TOC';
/* Read TOC to get the OPTIONS element descriptor */
opts=''
"EXECIO * DISKR TOC (OPEN FINIS STEM TOC."
do i = 1 to TOC.0;
  parse var TOC.i elem desc;
  if elem='OPTS:' then do;
    opts = desc;
    leave;
    end;
  end;
parse var opts o1 o2 o3 .
parse var o1 or1 ':' oc1; parse var o2 or2 ':' oc2;
/* Bring original external source into memory now, spliting into
 * three parts, Front (ahead of options), Back (after options)
 * and Middle (the options which will be replaced).
 */
"EXECIO 0 DISKR SQLVE (OPEN"
/* Front: all complete lines before options, into stem FR. */
FR.=''; FR.0=0;
if or1>1
  then "EXECIO" or1-1 "DISKR SQLVE (STEM FR.";
/* Middle: All records that have options on them
 * This will be at least one record (where options WOULD go).
 */
if o3='0'  /* no options were present */
  then "EXECIO      1      DISKR SQLVE (STEM MD.";
  else "EXECIO" 1+or2-or1 "DISKR SQLVE (STEM MD.";
/* Back: all the remaining complete lines */
BK.=''; BK.0=0;
"EXECIO * DISKR SQLVE (FINIS STEM BK.";
/* The Middle likely has portions of the Front, the Back, or both.
 * Separate those now, and then toss the middle. Process Back first.
 */
if FRm='' then FRm=''; /* collapse existing option indentation */
i=MD.0;
if o3='0'
  then j=oc1;      /* When no options, the BK middle starts at oc1. */
  else j=oc2+1;    /* With options, the BK middle starts after oc2. */
parse var MD.i MD.i =(j) BKm
if oc1<2
  then FRm='';
  else parse var MD.1 FRm =(oc1) MD.1

If MD.0 > 0 then do;
  say 'Removing these external SQL procedure options:'
  do i = 1 to MD.0; say MD.i; end;
  end;

drop MD. TOC.
/* We now have the external source in stems FR., BK.
 * and strings FRm and BKm. Release the old options.
 * Free our allocated data sets now. Current LISTING remains...
 */
'FREE DDNAME('allocList')';
allocList='';

/* Get the replacement options from the helper routine DSN8.DSN8EN1
 * Use the FUNCTION invocation of the SQLCALL service to obtain
 * the value of the last parameter (the SP output parm).
 */
call "OUTTRAP" 'TEMP.';
nat_opt=SQLCALL("DSN8.DSN8EN1('"sSch"', '"sNam"', VARCHAR('?', 5120))");
call "OUTTRAP" 'OFF';
if nat_opt='' | nat_opt='8' then do;
```

```
  msg='DSNTEJ67 Unable to obtain native options for replacement';
  call endWithListing msg ;
  end;
say 'Inserting these native SQL PL options:';
temp=nat_opt;
do while temp<>'';
  parse var temp opn '25'x temp;
  say opn;
  end;

/* Rewrite Original source using the new Native Options */
new_src='';
do i = 1 to FR.0; new_src=new_src || FR.i || '25'x; end; drop FR.
                  new_src=new_src || FRm              ; drop FRm
                  new_src=new_src || nat_opt          ; drop nat_opt
                  new_src=new_src || BKm              ; drop BKm
do i = 1 to BK.0; new_src=new_src || BK.i || '25'x; drop BK.i;  end;
call "STR2SQLV" new_src, 'DD:SQLV'
if oRecfm='F' then do;
     if oLrecl=80 then seq='SEQ'; else seq='';
     "SQLV2F" 'DD:SQLV' sourceFile seq
     end;
else "ANY2SQLV" 'DD:SQLV' sourceFile 'EXTEND'
"SQLV2F" 'DD:SQLV' 'DD:S80' 'S80'

chkoutSQLPL2:
/* Inspect the modified procedure source one last time
 * using the HOST(SQLPL) Checkout precompiler.
 */
'CHKSQLPL' 'DD:S80' 'DD:LISTING' '.' 'MAR(1,80)'
rrc=RC;
say 'Final HOST(SQLPL) Checkout Precompile ended with RC='rrc;
if rrc>0 then say 'Inspect the Listing for',
                  'additional SQLPL source coding issues';
'EXECIO * DISKR LISTING (OPEN FINIS STEM LISTING.';
call trimListing;
'EXECIO' listing.0 'DISKW LISTOUT (OPEN FINIS STEM LISTING.';
'FREE DDNAME(LISTING,SQLV,S80)';
exit rRC;
/* ------------------------------------------------------------- */

endWithListing:
  'EXECIO * DISKR LISTING (OPEN FINIS STEM LISTING.';
  call trimListing;
  'EXECIO' listing.0 'DISKW LISTOUT (OPEN FINIS STEM LISTING.';
  'FREE DDNAME(LISTING)';
  if arg(2,'E')
    then 'FREE DDNAME('arg(2)')'; /* other DDnames to free */
  if arg(1,'E')
    then say arg(1);   /* Message to end with */
  exit 8;

/* trimListing: Reduce the occurance of repeated headers, CC and page
 *              numbers in the listing, so it appears like one
 *              continuious stream.
 */
trimListing: procedure expose LISTING.
  hdrtypes = 'VERSION SYMBOL MESSAGES STATISTICS';
  j=0;  k=LISTING.0;  do i = 1 to k;
    parse var LISTING.i 1 cc +1 line;
    if cc='1' & "LEFT"(line,19)='DB2 SQL PRECOMPILER' then do; /*Hdr*/
      /* Reduce page header occurances */
      key="WORD"(line,4);
      loc="WORDPOS"(key,hdrtypes);
      if loc>0 then do;                        /* First hdr usage.   */
        parse var line line 'PAGE' .           /* Keep, w/o page num.*/
        hdrtypes = "DELWORD"(hdrtypes,loc,1);  /* Header now used.   */
        end;
      else iterate;                            /* Skip redundant hdr.*/
      end /*Hdr*/;
    j=j+1; LISTING.j="STRIP"(line,'T');
    end /* do i ... */;
  do i=j+1 to k; drop LISTING.i; end;
  LISTING.0=j;
  return 1+k-j; /* lines trimmed */

/* ------------------ end DSNTEJ67 ------------------------------- */
@@
./  ENDUP
//*********************************************************************
//*  Step 4:  Run the sample process to extract external SQL SP source
//*           from DB2, convert the source to native SQL PL, save the
```

```
//*           modified source in a data set and finish with a source
//*           listing written to the job log.
//*********************************************************************
//*
//PH067S04 EXEC PGM=IKJEFT01,DYNAMNBR=30
//SYSEXEC  DD DSN=&&REXXPDS,DISP=(OLD,PASS)
//SYSTSPRT DD SYSOUT=*
//DB2SSID  DD DSN=&&PARM0,DISP=(OLD,PASS)
//SPNAME   DD DSN=&&PARM1,DISP=(OLD,PASS)
//SOURCEDS DD DSN=&&PARM2,DISP=(OLD,PASS)
//LISTOUT  DD SYSOUT=*
//HELPRSP1 DD DSN=DSN!!0.SDSNIVPD(DSN8EN1),
//            DISP=SHR
//SYSTSIN  DD *
 %DSNTEJ67
/*
```

Related reference

"Sample programs to help you prepare and run external SQL procedures" on page 307
Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

Creating multiple versions of external procedures

For native SQL procedures, you can use Db2 to create and maintain multiple versions of the procedure. However, for external procedures including external SQL procedures, if you need multiple versions of a procedure, you need to maintain them manually.

Before you begin

Deprecated function: External SQL procedures are deprecated and not as fully supported as native SQL procedures. For best results, create native SQL procedures instead. For more information, see "Creating native SQL procedures" on page 231 and "Migrating an external SQL procedure to a native SQL procedure" on page 292.

Procedure

To create multiple versions of external procedures, including external SQL procedures, use one of the following techniques:

- Define multiple procedures with the same name in different schemas. You can subsequently use the SQL path to determine which version of the procedure is to be used by a calling program.
- Define multiple versions of the executable code. You can subsequently use a particular version by specifying the name of the load module for the version that you want to use on the EXTERNAL clause of the CREATE PROCEDURE statement or ALTER PROCEDURE statement.
- Define multiple packages for a procedure. You can subsequently use the COLLID option, the CURRENT PACKAGESET special register, or the CURRENT PACKAGE PATH special register to specify which version of the procedure is to be used by the calling application.
- Set up multiple WLM environments to use different versions of a procedure.

Adding and modifying data in tables from application programs

Your application program can add, modify, or delete data in any Db2 table for which you have the appropriate access.

Inserting data into tables

You can use several different methods to insert data into a table. Decide which method to use based on the amount of data that you need to insert and the other operations that your program needs to perform.

About this task

Besides using stand-alone INSERT statements, you can use the following ways to insert data into a table:

- You can user the MERGE statement to insert new data and update existing data in the same operation. .
- You can write an application program to prompt for and enter large amounts of data into a table.
- You can also use the Db2 LOAD utility to enter data from other sources.

Related tasks

Inserting data and updating data in a single operation
You can update existing data and insert new data in a single operation. This operation is useful when you want to update a table with a set of rows, some of which are changes to existing rows and some of which are new rows.

Related reference

LOAD (Db2 Utilities)

Inserting rows by using the INSERT statement

One way to insert data into tables is to use the SQL INSERT statement. This method is useful for inserting small amounts of data or inserting data from another table or view.

About this task

Procedure

Issue an INSERT statement.

By using INSERT statements, you can do the following actions:

- Specify the column values to insert a single row. You can specify constants, host variables, expressions, DEFAULT, or NULL by using the VALUES clause.
- In an application program, specify arrays of column values to insert multiple rows into a table. Use host-variable arrays in the VALUES clause of the INSERT FOR *n* ROWS statement to add multiple rows of column values to a table.
- Include a SELECT statement in the INSERT statement to tell Db2 that another table or view contains the data for the new row or rows.

In each case, for every row that you insert, you must provide a value for any column that does not have a default value. For a column that meets one of the following conditions, specify DEFAULT to tell Db2 to insert the default value for that column:

- The column is nullable.
- The column is defined with a default value.
- The column has data type ROWID. ROWID columns always have default values.
- The column is an identity column. Identity columns always have default values.
- The column is a row change timestamp column.

The values that you can insert into a ROWID column, an identity column, or a row change timestamp column depend on whether the column is defined with GENERATED ALWAYS or GENERATED BY DEFAULT.

You can use the VALUES clause of the INSERT statement to insert a single row of column values into a table. You can either name all of the columns for which you are providing values, or you can omit the list of column names. If you omit the column name list, you must specify values for **all** of the columns.

Recommendation: For static INSERT statements, name all of the columns for which you are providing values for the following reasons:

- Your INSERT statement is independent of the table format. (For example, you do not need to change the statement when a column is added to the table.)
- You can verify that you are specifying the values in order.
- Your source statements are more self-descriptive.

If you do not name the columns in a static INSERT statement, and a column is added to the table, an error can occur if the INSERT statement is rebound. An error will occur after any rebind of the INSERT statement unless you change the INSERT statement to include a value for the new column. This is true even if the new column has a default value.

When you list the column names, you must specify their corresponding values in the same order as in the list of column names.

Examples

Example

The following statement inserts information about a new department into the YDEPT table.

```
INSERT INTO YDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
  VALUES ('E31', 'DOCUMENTATION', '000010', 'E01', ' ');
```

After inserting a new department row into your YDEPT table, you can use a SELECT statement to see what you have loaded into the table. The following SQL statement shows you all of the new department rows that you have inserted:

```
SELECT *
  FROM YDEPT
  WHERE DEPTNO LIKE 'E%'
  ORDER BY DEPTNO;
```

The result table looks similar to the following output:

```
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT  LOCATION
======  ===================================  ======  ========  ===========
E01     SUPPORT SERVICES                     000050  A00       -----------
E11     OPERATIONS                           000090  E01       -----------
E21     SOFTWARE SUPPORT                     000100  E01       -----------
E31     DOCUMENTATION                        000010  E01       -----------
```

Example

The following statement inserts information about a new employee into the YEMP table. Because the WORKDEPT column is a foreign key, the value that is inserted for that column (E31) must be a value in the primary key column, which is DEPTNO in the YDEPT table.

```
INSERT INTO YEMP
  VALUES ('000400', 'RUTHERFORD', 'B', 'HAYES', 'E31', '5678', '1998-01-01',
          'MANAGER', 16, 'M', '1970-07-10', 24000, 500, 1900);
```

Example

The following statement also inserts a row into the YEMP table. Because the unspecified columns allow null values, Db2 inserts null values into the columns that you do not specify.

```
INSERT INTO YEMP
  (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, JOB)
  VALUES ('000410', 'MILLARD', 'K', 'FILLMORE', 'D11', '4888', 'MANAGER');
```

Related concepts

Rules for inserting data into an identity column
An *identity column* contains a unique numeric value for each row in the table. Whether you can insert data into an identity column and how that data gets inserted depends on how the column is defined.

Rules for inserting data into a ROWID column
A *ROWID column* contains unique values that identify each row in a table. Whether you can insert data into a ROWID column and how that data gets inserted depends on how the column is defined.

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Inserting rows into a table from another table
You can copy one or more rows from one table into another table.

Related reference

INSERT (Db2 SQL)

CREATE TABLE (Db2 SQL)

Inserting rows into a table from another table

You can copy one or more rows from one table into another table.

Procedure

Use a fullselect within an INSERT statement.

Examples

Example

The following SQL statement creates a table named TELE:

```
CREATE TABLE TELE
  (NAME2  VARCHAR(15)  NOT NULL,
   NAME1  VARCHAR(12)  NOT NULL,
   PHONE  CHAR(4));
```

The following statement copies data from DSN8B10.EMP into the newly created table:

```
INSERT INTO TELE
  SELECT LASTNAME, FIRSTNME, PHONENO
    FROM DSN8B10.EMP
    WHERE WORKDEPT = 'D21';
```

The two previous statements create and fill a table, TELE, that looks similar to the following table:

```
NAME2            NAME1          PHONE
===============  ============   =====
PULASKI          EVA            7831
JEFFERSON        JAMES          2094
MARINO           SALVATORE      3780
SMITH            DANIEL         0961
JOHNSON          SYBIL          8953
PEREZ            MARIA          9001
MONTEVERDE       ROBERT         3780
```

The CREATE TABLE statement example creates a table which, at first, is empty. The table has columns for last names, first names, and phone numbers, but does not have any rows.

The INSERT statement fills the newly created table with data that is selected from the DSN8B10.EMP table: the names and phone numbers of employees in department D21.

Example

The following CREATE statement creates a table that contains an employee's department name and phone number. The fullselect within the INSERT statement fills the DLIST table with data from rows that are selected from two existing tables, DSN8B10.DEPT and DSN8B10.EMP.

```
CREATE TABLE DLIST
  (DEPT    CHAR(3)      NOT NULL,
   DNAME   VARCHAR(36)          ,
   LNAME   VARCHAR(15)  NOT NULL,
   FNAME   VARCHAR(12)  NOT NULL,
   INIT    CHAR                 ,
   PHONE   CHAR(4) );
```

```
INSERT INTO DLIST
  SELECT DEPTNO, DEPTNAME, LASTNAME, FIRSTNME, MIDINIT, PHONENO
    FROM DSN8B10.DEPT, DSN8B10.EMP
    WHERE DEPTNO = WORKDEPT;
```

Rules for inserting data into a ROWID column

A *ROWID column* contains unique values that identify each row in a table. Whether you can insert data into a ROWID column and how that data gets inserted depends on how the column is defined.

A *ROWID column* is a column that is defined with a ROWID data type. You must have a column with a ROWID data type in a table that contains a LOB column. The ROWID column is stored in the base table and is used to look up the actual LOB data in the LOB table space. In addition, a ROWID column enables you to write queries that navigate directly to a row in a table. For information about using ROWID columns for direct-row access, see "Specifying direct row access by using row IDs" on page 430.

Before you insert data into a ROWID column, you must know how the ROWID column is defined. ROWID columns can be defined as GENERATED ALWAYS or GENERATED BY DEFAULT. GENERATED ALWAYS means that Db2 generates a value for the column, and you cannot insert data into that column. If the column is defined as GENERATED BY DEFAULT, you can insert a value, and Db2 provides a default value if you do not supply one.

Example: Suppose that tables T1 and T2 have two columns: an integer column and a ROWID column. For the following statement to run successfully, ROWIDCOL2 must be defined as GENERATED BY DEFAULT.

```
INSERT INTO T2 (INTCOL2,ROWIDCOL2)
  SELECT * FROM T1;
```

If ROWIDCOL2 is defined as GENERATED ALWAYS, you cannot insert the ROWID column data from T1 into T2, but you can insert the integer column data. To insert only the integer data, use one of the following methods:

- Specify only the integer column in your INSERT statement, as in the following statement:

  ```
  INSERT INTO T2 (INTCOL2)
    SELECT INTCOL1 FROM T1;
  ```

- Specify the OVERRIDING USER VALUE clause in your INSERT statement to tell Db2 to ignore any values that you supply for system-generated columns, as in the following statement:

  ```
  INSERT INTO T2 (INTCOL2,ROWIDCOL2) OVERRIDING USER VALUE
    SELECT * FROM T1;
  ```

Related concepts

Direct row access (PRIMARY_ACCESSTYPE='D') (Db2 Performance)

ROWID data type (Introduction to Db2 for z/OS)

Related tasks

Specifying direct row access by using row IDs

For some applications, you can use the value of a ROWID column to navigate directly to a row.

Rules for inserting data into an identity column

An *identity column* contains a unique numeric value for each row in the table. Whether you can insert data into an identity column and how that data gets inserted depends on how the column is defined.

An *identity column* is a numeric column, defined in a CREATE TABLE or ALTER TABLE statement, that has ascending or descending values. For an identity column to be as useful as possible, its values should also be unique. The column has a SMALLINT, INTEGER, or DECIMAL(*p*,0) data type and is defined with the AS IDENTITY clause. The AS IDENTITY clause specifies that the column is an identity column. For information about using identity columns to uniquely identify rows, see "Identity columns" on page 126

Before you insert data into an identity column, you must know how the column is defined. Identity columns are defined with the GENERATED ALWAYS or GENERATED BY DEFAULT clause. GENERATED ALWAYS means that Db2 generates a value for the column, and you cannot insert data into that column. If the column is defined as GENERATED BY DEFAULT, you can insert a value, and Db2 provides a default value if you do not supply one.

Example: Suppose that tables T1 and T2 have two columns: a character column and an integer column that is defined as an identity column. For the following statement to run successfully, IDENTCOL2 must be defined as GENERATED BY DEFAULT.

```
INSERT INTO T2 (CHARCOL2,IDENTCOL2)
  SELECT * FROM T1;
```

If IDENTCOL2 is defined as GENERATED ALWAYS, you cannot insert the identity column data from T1 into T2, but you can insert the character column data. To insert only the character data, use one of the following methods:

- Specify only the character column in your INSERT statement, as in the following statement:

  ```
  INSERT INTO T2 (CHARCOL2)
    SELECT CHARCOL1 FROM T1;
  ```

- Specify the OVERRIDING USER VALUE clause in your INSERT statement to tell Db2 to ignore any values that you supply for system-generated columns, as in the following statement:

  ```
  INSERT INTO T2 (CHARCOL2,IDENTCOL2) OVERRIDING USER VALUE
    SELECT * FROM T1;
  ```

Restrictions when assigning values to columns with distinct types

Certain conditions are required when you assign a column value to another column or when you assign a constant to a column of a distinct type. If the conditions are not met, you cannot assign the value.

When assigning a column value to another column or a constant to a column of a distinct type, the type of the value that is to be assigned must match the column type, or you must be able to cast one type to the other. Otherwise, you cannot assign the value.

If you need to assign a value of one distinct type to a column of another distinct type, a function must exist that converts the value from one type to another. Because Db2 provides cast functions only between distinct types and their source types, you must write the function to convert from one distinct type to another.

Assigning column values to columns with different distinct types

Suppose tables JAPAN_SALES and JAPAN_SALES_03 are defined like this:

```
CREATE TABLE JAPAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1990),
   TOTAL         JAPANESE_YEN);

CREATE TABLE JAPAN_SALES_03
```

```
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR);
```

You need to insert values from the TOTAL column in JAPAN_SALES into the TOTAL column of JAPAN_SALES_03. Because INSERT statements follow assignment rules, Db2 does not let you insert the values directly from one column to the other because the columns are of different distinct types. Suppose that a user-defined function called US_DOLLAR has been written that accepts values of type JAPANESE_YEN as input and returns values of type US_DOLLAR. You can then use this function to insert values into the JAPAN_SALES_03 table:

```
INSERT INTO JAPAN_SALES_03
  SELECT PRODUCT_ITEM, US_DOLLAR(TOTAL)
  FROM JAPAN_SALES
  WHERE YEAR = 2003;
```

Assigning column values with distinct types to host variables

The rules for assigning distinct types to host variables or host variables to columns of distinct types differ from the rules for constants and columns.

You can assign a column value of a distinct type to a host variable if you can assign a column value of the distinct type's source type to the host variable. In the following example, you can assign SIZECOL1 and SIZECOL2, which has distinct type SIZE, to host variables of type double and short because the source type of SIZE, which is INTEGER, can be assigned to host variables of type double or short.

```
EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short  hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
⋮
SELECT SIZECOL1, SIZECOL2
  INTO :hv1, :hv2
  FROM TABLE1;
```

Assigning host variable values to columns with distinct types

When you assign a value in a host variable to a column with a distinct type, the type of the host variable must be able to cast to the distinct type.

In this example, values of host variable hv2 can be assigned to columns SIZECOL1 and SIZECOL2, because C data type short is equivalent to Db2 data type SMALLINT, and SMALLINT is promotable to data type INTEGER. However, values of hv1 cannot be assigned to SIZECOL1 and SIZECOL2, because C data type double, which is equivalent to Db2 data type DOUBLE, is not promotable to data type INTEGER.

```
EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short  hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
⋮
INSERT INTO TABLE1
  VALUES (:hv1,:hv1);     /* Invalid statement */
INSERT INTO TABLE1
  VALUES (:hv2,:hv2);     /* Valid statement   */
```

Related concepts

Promotion of data types (Db2 SQL)

Inserting data and updating data in a single operation

You can update existing data and insert new data in a single operation. This operation is useful when you want to update a table with a set of rows, some of which are changes to existing rows and some of which are new rows.

About this task

You can update existing data and insert new data in a single operation by using the MERGE statement.

For example, an application might request a set of rows from a database, enable a user to modify the data through a GUI, and then store the modified data in the database. Some of this modified data is updates to existing rows, and some of this data is new rows. You can do these update and insert operations in one step.

Procedure

Issue a MERGE statement.

To update existing data and inserting new data, specify a MERGE statement with the WHEN MATCHED and WHEN NOT MATCHED clauses. These clauses specify how Db2 handles matched and unmatched data. If Db2 finds a matching row, that row is updated. If Db2 does not find a matching row, a new row is inserted.

Example

Suppose that you need to update the inventory at a car dealership. You need to add new car models to the inventory and update information about car models that are already in the inventory.

You could make these changes with the following series of statements:

```
UPDATE INVENTORY
  SET QUANTITY = QUANTITY + :hv_delta
  WHERE MODEL = :hv_model;

--begin pseudo code
if sqlcode >= 0
then do
       GD
       if rc = 0  then INSERT..
     end
-- end pseudo code

GET DIAGNOSTICS :rc = ROW_COUNT;

IF rc = 0 THEN
INSERT INTO INVENTORY VALUES (:hv_model, :hv_delta);
END IF;
```

The MERGE statement simplifies the update and the insert into a single statement:

```
MERGE INTO INVENTORY
USING ( VALUES (:hv_model, :hv_delta) ) AS SOURCE(MODEL, DELTA)
ON INVENTORY.MODEL = SOURCE.MODEL
  WHEN MATCHED THEN UPDATE SET QUANTITY = QUANTITY + SOURCE.DELTA
  WHEN NOT MATCHED THEN INSERT VALUES (SOURCE.MODEL, SOURCE.DELTA)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

Related reference

MERGE (Db2 SQL)

Selecting values while merging data

When you update existing data and insert new data in a single merge operation, you can select values from those rows at the same time.

Procedure

Specifying the MERGE statement in the FROM clause of the SELECT statement.

When you merge one or more rows into a table, you can retrieve:

- The value of an automatically generated column such as a ROWID or identity column
- Any default values for columns
- All values for a merged row, without specifying individual column names
- Calculated values based on the changes to merged rows

Specify the FINAL TABLE clause with SELECT FROM MERGE statements. The FINAL TABLE consists of the rows of the table or view after the merge occurs.

Example

Suppose that you need to input data into the STOCK table, which contains company stock symbols and stock prices from your stock portfolio. Some of your input data refers to companies that are already in the STOCK table; some of the data refers to companies that you are adding to your stock portfolio. If the stock symbol exists in the SYMBOL column of the STOCK table, you need to update the PRICE column. If the company stock symbol is not yet in the STOCK table, you need to insert a new row with the stock symbol and the stock price. Furthermore, you need to add a new value DELTA to your output to show the change in stock price.

Suppose that the STOCK table contains the data that is shown in Table 63 on page 342.

Table 63. STOCK table before SELECT FROM MERGE statement

SYMBOL	PRICE
XCOM	95.00
YCOM	24.50

Now, suppose that :hv_symbol and :hv_price are host-variable arrays that contain updated data that corresponds to the data that is shown in Table 63 on page 342. Table 64 on page 342 shows the host variable data for stock activity.

Table 64. Host-variable arrays of stock activity

hv_symbol	hv_price
XCOM	97.00
NEWC	30.00
XCOM	107.00

NEWC is new to the STOCK table, so its symbol and price need to be inserted into the STOCK table. The rows for XCOM in Table 64 on page 342represent changed stock prices, so these values need to be updated in the STOCK table. Also, the output needs to show the change in stock prices as a DELTA value.

The following SELECT FROM MERGE statement updates the price of XCOM, inserts the symbol and price for NEWC, and returns an output that includes a DELTA value for the change in stock price.

```
SELECT SYMBOL, PRICE, DELTA FROM FINAL TABLE
  (MERGE INTO STOCK AS S INCLUDE (DELTA DECIMAL(5,20)
   USING ((:hv_symbol, :hv_price) FOR :hv_nrows ROWS) AS R (SYMBOL, PRICE)
   ON S.SYMBOL = R.SYMBOL
     WHEN MATCHED THEN UPDATE SET
       DELTA = R.PRICE - S.PRICE, PRICE=R.PRICE
     WHEN NOT MATCHED THEN INSERT
       (SYMBOL, PRICE, DELTA) VALUES (R.SYMBOL, R.PRICE, R.PRICE)
NOT ATOMIC CONTINUE ON SQLEXCEPTION);
```

The INCLUDE clause specifies that an additional column, DELTA, can be returned in the output without adding a column to the STOCK table. The UPDATE portion of the MERGE statement sets the DELTA value to the differential of the previous stock price with the value set for the update operation. The INSERT portion of the MERGE statement sets the DELTA value to the same value as the PRICE column.

After the SELECT FROM MERGE statement is processed, the STOCK table contains the data that is shown in Table 65 on page 343.

Table 65. STOCK table after SELECT FROM MERGE statement	
SYMBOL	**PRICE**
XCOM	107.00
YCOM	24.50
NEWC	30.00

The following output of the SELECT FROM MERGE statement includes both updates to XCOM and a DELTA value for each output row.

```
SYMBOL    PRICE      DELTA
=============================
XCOM       97.00      2.00
NEWC       30.00     30.00
XCOM      107.00     10.00
```

Selecting values while inserting data

When you insert rows into a table, you can also select values from the inserted rows at the same time.

About this task

When you insert one or more new rows into a table, you can also retrieve rows, including the following values:

- The value of an automatically generated column such as a ROWID or identity column
- Any default values for columns
- All values for an inserted row, without specifying individual column names
- All values that are inserted by a multiple-row INSERT operation
- Values that are changed by a BEFORE INSERT trigger

Procedure

Specify the INSERT statement in the FROM clause of the SELECT statement.

The rows that are inserted into the target table produce a result table whose columns can be referenced in the SELECT list of the query. The columns of the result table are affected by the columns, constraints, and triggers that are defined for the target table:

- The result table includes Db2-generated values for identity columns, ROWID columns, or row change timestamp columns.
- Before Db2 generates the result table, it enforces any constraints that affect the insert operation (that is, check constraints, unique index constraints, and referential integrity constraints).
- The result table includes any changes that result from a BEFORE trigger that is activated by the insert operation. An AFTER trigger does not affect the values in the result table.

Examples

In addition to examples that use the Db2 sample tables, the examples in this topic use an EMPSAMP table that has the following definition:

```
CREATE TABLE EMPSAMP
  (EMPNO     INTEGER GENERATED ALWAYS AS IDENTITY,
   NAME      CHAR(30),
   SALARY    DECIMAL(10,2),
   DEPTNO    SMALLINT,
   LEVEL     CHAR(30),
```

```
HIRETYPE  VARCHAR(30) NOT NULL WITH DEFAULT 'New Hire',
HIREDATE  DATE NOT NULL WITH DEFAULT);
```

Example 1: Retrieving generated column values

Assume that you need to insert a row for a new employee into the EMPSAMP table. To find out the values for the generated EMPNO, HIRETYPE, and HIREDATE columns, use the following SELECT FROM INSERT statement:

```
SELECT EMPNO, HIRETYPE, HIREDATE
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, DEPTNO, LEVEL)
                    VALUES('Mary Smith', 35000.00, 11, 'Associate'));
```

The SELECT statement returns the Db2-generated identity value for the EMPNO column, the default value 'New Hire' for the HIRETYPE column, and the value of the CURRENT DATE special register for the HIREDATE column.

Recommendation: Use the SELECT FROM INSERT statement to insert a row into a parent table and retrieve the value of a primary key that was generated by Db2 (a ROWID or identity column). In another INSERT statement, specify this generated value as a value for a foreign key in a dependent table.

Example 2: Retrieving values updated by triggers

Suppose that a BEFORE INSERT trigger is created on table EMPSAMP to give all new employees at the Associate level a $5000 increase in salary. The trigger has the following definition:

```
CREATE TRIGGER NEW_ASSOC
  NO CASCADE BEFORE INSERT ON EMPSAMP
  REFERENCING NEW AS NEWSALARY
  FOR EACH ROW MODE DB2SQL
    WHEN (NEWSALARY.LEVEL = 'ASSOCIATE')
    BEGIN ATOMIC
      SET NEWSALARY.SALARY = NEWSALARY.SALARY + 5000.00;
    END;
```

The INSERT statement in the FROM clause of the following SELECT statement inserts a new employee into the EMPSAMP table:

```
SELECT NAME, SALARY
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, LEVEL)
                    VALUES('Mary Smith', 35000.00, 'Associate'));
```

The SELECT statement returns a salary of 40000.00 for Mary Smith instead of the initial salary of 35000.00 that was explicitly specified in the INSERT statement.

Selecting values when you insert a single row:

When you insert a new row into a table, you can retrieve any column in the result table of the SELECT FROM INSERT statement. When you embed this statement in an application, you retrieve the row into host variables by using the SELECT ... INTO form of the statement.

Example 4: Retrieving all values for a row inserted intro a structure.

You can retrieve all the values for a row that is inserted into a structure. For example, in the following statement :empstruct is a host variable structure that is declared with variables for each of the columns in the EMPSAMP table.

```
EXEC SQL SELECT * INTO :empstruct
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, DEPTNO, LEVEL)
                    VALUES('Mary Smith', 35000.00, 11, 'Associate'));
```

Example 4: Selecting values when inserting data into a view

If the INSERT statement references a view that is defined with a search condition, that view must be defined with the WITH CASCADED CHECK OPTION option. When you insert data into the view, the result table of the SELECT FROM INSERT statement includes only rows that satisfy the view definition.

Because view V1 is defined with the WITH CASCADED CHECK OPTION option, you can reference V1 in the INSERT statement:

```
CREATE VIEW V1 AS
  SELECT C1, I1 FROM T1 WHERE I1 > 10
  WITH CASCADED CHECK OPTION;

SELECT C1 FROM
  FINAL TABLE (INSERT INTO V1 (I1) VALUES(12));
```

The value 12 satisfies the search condition of the view definition, and the result table consists of the value for C1 in the inserted row.

If you use a value that does not satisfy the search condition of the view definition, the insert operation fails, and Db2 returns an error.

Example 5: Selecting ROWID values when inserting multiple rows

In an application program, to retrieve values from the insertion of multiple rows, declare a cursor so that the INSERT statement is in the FROM clause of the SELECT statement of the cursor.

To see the values of the ROWID columns that are inserted into the employee photo and resume table, you can declare the following cursor:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8B10.EMP_PHOTO_RESUME (EMPNO)
                    SELECT EMPNO FROM DSN8B10.EMP);
```

Example 6: Using the FETCH FIRST clause

To see only the first five rows that are inserted into the employee photo and resume table, use the FETCH FIRST clause:

```
EXEC SQL DECLARE CS2 CURSOR FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8B10.EMP_PHOTO_RESUME (EMPNO)
                    SELECT EMPNO FROM DSN8B10.EMP)
  FETCH FIRST 5 ROWS ONLY;
```

Example 7: Using the INPUT SEQUENCE clause

To retrieve rows in the order in which they are inserted, use the INPUT SEQUENCE clause:

```
EXEC SQL DECLARE CS3 CURSOR FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8B10.EMP_PHOTO_RESUME (EMPNO)
                    VALUES(:hva_empno)
                    FOR 5 ROWS)
  ORDER BY INPUT SEQUENCE;
```

The INPUT SEQUENCE clause can be specified only if an INSERT statement is in the FROM clause of the SELECT statement. In this example, the rows are inserted from an array of employee numbers.

Example 8: Inserting rows with multiple encoding CCSIDs

Suppose that you want to populate an ASCII table with values from an EBCDIC table and then see selected values from the ASCII table. You can use the following cursor to select the EBCDIC columns, populate the ASCII table, and then retrieve the ASCII values:

```
EXEC SQL DECLARE CS4 CURSOR FOR
  SELECT C1, C2
  FROM FINAL TABLE (INSERT INTO ASCII_TABLE
                    SELECT * FROM EBCDIC_TABLE);
```

Example 9: Selecting additional columns when inserting data

You can use the INCLUDE clause to introduce a new column to the result table but not add a column to the target table.

Suppose that you need to insert department number data into the project table. Suppose also, that you want to retrieve the department number and the corresponding manager number for each department. Because MGRNO is not a column in the project table, you can use the INCLUDE clause to include the manager number in your result but not in the insert operation. The following SELECT FROM INSERT statement performs the insert operation and retrieves the data.

```
DECLARE CS1 CURSOR FOR
 SELECT manager_num, projname FROM FINAL TABLE
   (INSERT INTO PROJ (DEPTNO) INCLUDE(manager_num CHAR(6))
      SELECT DEPTNO, MGRNO FROM DEPT);
```

Example 10: Result table of the cursor when you insert multiple rows

In an application program, when you insert multiple rows into a table, you declare a cursor so that the INSERT statement is in the FROM clause of the SELECT statement of the cursor. The result table of the cursor is determined during OPEN cursor processing. The result table may or may not be affected by other processes in your application.

When you declare a scrollable cursor, the cursor must be declared with the INSENSITIVE keyword if an INSERT statement is in the FROM clause of the cursor specification. The result table is generated during OPEN cursor processing and does not reflect any future changes. You cannot declare the cursor with the SENSITIVE DYNAMIC or SENSITIVE STATIC keywords.

When you declare a non-scrollable cursor, any searched updates or deletes do not affect the result table of the cursor. The rows of the result table are determined during OPEN cursor processing.

For example, assume that your application declares a cursor, opens the cursor, performs a fetch, updates the table, and then fetches additional rows:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT SALARY
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, LEVEL)
                    SELECT NAME, INCOME, BAND FROM OLD_EMPLOYEE);
EXEC SQL OPEN CS1;
EXEC SQL FETCH CS1 INTO :hv_salary;
/* print fetch result */
...
EXEC SQL UPDATE EMPSAMP SET SALARY = SALARY + 500;
while (SQLCODE == 0) {
  EXEC SQL FETCH CS1 INTO :hv_salary;
  /* print fetch result */
  ...
}
```

The fetches that occur after the updates return the rows that were generated when the cursor was opened. If you use a simple SELECT (with no INSERT statement in the FROM clause), the fetches might return the updated values, depending on the access path that Db2 uses.

Example 11: Effect of WITH HOLD

When you declare a cursor with the WITH HOLD option and open the cursor, all of the rows are inserted into the target table. The WITH HOLD option has no effect on the SELECT FROM INSERT statement of the cursor definition. After your application performs a commit, you can continue to retrieve all of the inserted rows.

Assume that the employee table in the Db2 sample application has five rows. Your application declares a WITH HOLD cursor, opens the cursor, fetches two rows, performs a commit, and then fetches the third row successfully:

```
EXEC SQL DECLARE CS2 CURSOR WITH HOLD FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8B10.EMP_PHOTO_RESUME (EMPNO)
                    SELECT EMPNO FROM DSN8B10.EMP);
EXEC SQL OPEN CS2;                              /* Inserts 5 rows */
EXEC SQL FETCH CS2 INTO :hv_rowid;   /* Retrieves ROWID for 1st row */
EXEC SQL FETCH CS2 INTO :hv_rowid;   /* Retrieves ROWID for 2nd row */
EXEC SQL COMMIT;                                /* Commits 5 rows */
EXEC SQL FETCH CS2 INTO :hv_rowid;   /* Retrieves ROWID for 3rd row */
```

Example 12: Effect of SAVEPOINT and ROLLBACK

A savepoint is a point in time within a unit of recovery to which relational database changes can be rolled back. You can set a savepoint with the SAVEPOINT statement.

When you set a savepoint prior to opening the cursor and then roll back to that savepoint, all of the insertions are undone.

Assume that your application declares a cursor, sets a savepoint, opens the cursor, sets another savepoint, rolls back to the second savepoint, and then rolls back to the first savepoint:

```
EXEC SQL DECLARE CS3 CURSOR FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8B10.EMP_PHOTO_RESUME (EMPNO)
                    SELECT EMPNO FROM DSN8B10.EMP);
EXEC SQL SAVEPOINT A ON ROLLBACK RETAIN CURSORS;       /* Sets 1st savepoint */
EXEC SQL OPEN CS3;
EXEC SQL SAVEPOINT B ON ROLLBACK RETAIN CURSORS;       /* Sets 2nd savepoint */
...
EXEC SQL ROLLBACK TO SAVEPOINT B;  /* Rows still in DSN8B10.EMP_PHOTO_RESUME */
...
EXEC SQL ROLLBACK TO SAVEPOINT A;            /* All inserted rows are undone */
```

Example 13: Errors during SELECT INTO processing

In an application program, when you insert one or more rows into a table by using the SELECT FROM INSERT statement, the result table of the insert operation may or may not be affected, depending on where the error occurred in the application processing.

If the insert processing or the select processing fails during a SELECT INTO statement, no rows are inserted into the target table, and no rows are returned from the result table of the insert operation. For example, assume that the employee table of the Db2 sample application has one row, and that the SALARY column has a value of 9999000.00.

```
EXEC SQL SELECT EMPNO INTO :hv_empno
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY)
                    SELECT FIRSTNAME || MIDINIT || LASTNAME,
                           SALARY + 10000.00
                    FROM DSN8B10.EMP)
```

The addition of 10000.00 causes a decimal overflow to occur, and no rows are inserted into the EMPSAMP table.

Example 14: Errors during OPEN cursor processing

If the insertion of any row fails during the OPEN cursor processing, all previously successful insertions are undone. The result table of the insert is empty.

Example 15: Errors during FETCH processing

If the FETCH statement fails while retrieving rows from the result table of the insert operation, a negative SQLCODE is returned to the application, but the result table still contains the original number of rows that was determined during the OPEN cursor processing. At this point, you can undo all of the inserts.

Assume that the result table contains 100 rows and the 90th row that is being fetched from the cursor returns a negative SQLCODE:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT EMPNO
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY)
                    SELECT FIRSTNAME || MIDINIT || LASTNAME, SALARY + 10000.00
                      FROM DSN8B10.EMP);
EXEC SQL OPEN CS1;                                 /* Inserts 100 rows */
while (SQLCODE == 0)
  EXEC SQL FETCH CS1 INTO :hv_empno;
if (SQLCODE == -904)           /* If SQLCODE is -904, undo all inserts */
  EXEC SQL ROLLBACK;
else                                           /* Else, commit inserts */
  EXEC SQL COMMIT;
```

Related concepts

Held and non-held cursors

A held cursor does not close after a commit operation. A cursor that is not held closes after a commit operation. You specify whether you want a cursor to be held or not held by including or omitting the WITH HOLD clause when you declare the cursor.

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Identity columns
An identity column contains a unique numeric value for each row in the table. Db2 can automatically generate sequential numeric values for this column as rows are inserted into the table. Thus, identity columns are ideal for primary key values, such as employee numbers or product numbers.

Types of cursors
You can declare row-positioned or rowset-positioned cursors in a number of ways. These cursors can be scrollable or not scrollable, held or not held, or returnable or not returnable.

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Retrieving a set of rows by using a cursor
In an application program, you can retrieve a set of rows from a table or a result table that is returned by a stored procedure. You can retrieve one or more rows at a time.

Undoing selected changes within a unit of work by using savepoints
Savepoints enable you to undo selected changes within a unit of work. Your application can set any number of savepoints and then specify a specific savepoint to indicate which changes to undo within the unit of work.

Related reference

Command line processor BIND command
Use the command line processor BIND command to bind DBRMs that are in z/OS UNIX HFS files to packages.

Preserving the order of a derived table

When you specify SELECT FROM INSERT, SELECT FROM UPDATE, SELECT FROM DELETE, or SELECT FROM MERGE, you can preserve the order of the derived table. This action ensures that the result rows of a fullselect follow the same order as the result table of a subquery within the fullselect.

Procedure

To preserve the order of the derived table specify the ORDER OF clause with the ORDER BY clause.

These two clauses ensure that the result rows of a fullselect follow the same order as the result table of a subquery within the fullselect.

You can use the ORDER OF clause in any query that uses an ORDER BY clause, but the ORDER OF clause is most useful with queries that contain a set operator, such as UNION.

Examples

Example

The following example retrieves the following rows:

- Rows of table T1 in no specified order
- Rows of table T2 in the order of the first column in table T2

The example query then performs a UNION ALL operation on the results of the two subqueries. The ORDER BY ORDER OF UTABLE clause in the query specifies that the fullselect result rows are to be returned in the same order as the result rows of the UNION ALL statement.

```
SELECT * FROM
  (SELECT * FROM T1
     UNION ALL
   (SELECT * FROM T2 ORDER BY 1)
  ) AS UTABLE
ORDER BY ORDER OF UTABLE;
```

Example

The following example joins data from table T1 to the result table of a nested table expression. The nested table expression is ordered by the second column in table T2. The ORDER BY ORDER OF TEMP clause in the query specifies that the fullselect result rows are to be returned in the same order as the nested table expression.

```
SELECT T1.C1, T1.C2, TEMP.Cy, TEMP.Cx
FROM T1, (SELECT T2.C1, T2.C2 FROM T2 ORDER BY 2) as TEMP(Cx, Cy)
WHERE Cy = T1.C1
ORDER BY ORDER OF TEMP;
```

Alternatively, you can produce the same result by explicitly stating the ORDER BY column TEMP.Cy in the fullselect instead of using the ORDER OF syntax.

```
SELECT T1.C1, T1.C2, TEMP.Cy, TEMP.Cx
FROM T1, (SELECT T2.C1, T2.C2 FROM T2 ORDER BY 2) as TEMP(Cx, Cy)
WHERE Cy = T1.C1
ORDER BY TEMP.Cy;
```

Related reference

fullselect (Db2 SQL)

order-by-clause (Db2 SQL)

Adding data to the end of a table

In a relational database, the rows of a table are not ordered, and thus, the table has no "end." However, depending on your goal, you can perform several actions to simulate adding data to the end of a table.

About this task

Question: How can I add data to the end of a table?

Answer: Though the question is often asked, it has no meaning in a relational database. The rows of a base table are not ordered; hence, the table does not have an "end".

However, depending on your goal, you can perform one of the following actions to simulate adding data to the end of a table:

- If your goal is to get a result table that is ordered according to when the rows were inserted, define a unique index on a TIMESTAMP column in the table definition. Then, when you retrieve data from the table, use an ORDER BY clause that names that column. The newest insert appears last.
- If your goal is for Db2 to insert rows in the next available free space, without preserving clustering order, specify the APPEND YES option when you create or alter the table. Specifying this option might reduce the time it takes to insert rows, because Db2 does not spend time searching for free space.

Storing data that does not have a tabular format

Db2 provides several options for you to store large volumes of data that is not defined as a set of columns in a table.

About this task

Question: How can I store a large volume of data that is not defined as a set of columns in a table?

Answer: You can store the data in a table in a binary string, a LOB, or an XML column.

Updating table data

You can change a column value to another value or remove the column value altogether.

Procedure

To change the data in a table, use the UPDATE statement.
For example, suppose that an employee relocates. To update several items of the employee's data in the YEMP work table to reflect the move, you can execute the following statement:

```
UPDATE YEMP
  SET JOB = 'MANAGER ',
  PHONENO ='5678'
  WHERE EMPNO = '000400';
```

You can also use the UPDATE statement to remove a value from a column (without removing the row) by changing the column value to null.

You cannot update rows in a created temporary table, but you can update rows in a declared temporary table.

The SET clause names the columns that you want to update and provides the values that you want to assign to those columns. You can replace a column value in the SET clause with any of the following items:

- A null value

  The column to which you assign the null value must not be defined as NOT NULL.
- An expression, which can be any of the following items:
  - A column
  - A constant
  - A scalar fullselect
  - A host variable
  - A special register
- A default value

  If you specify DEFAULT, Db2 determines the value based on how the corresponding column is defined in the table.

In addition, you can replace one or more column values in the SET clause with the column values in a row that is returned by a fullselect.

Next, identify the rows to update:

- To update a single row, use a WHERE clause that locates one, and only one, row.
- To update several rows, use a WHERE clause that locates only the rows that you want to update.

If you omit the WHERE clause, Db2 updates **every row** in the table or view with the values that you supply.

If Db2 finds an error while executing your UPDATE statement (for example, an update value that is too large for the column), it stops updating and returns an error. No rows in the table change. Rows that were already changed, if any, are restored to their previous values. If the UPDATE statement is successful, SQLERRD(3) is set to the number of rows that are updated.

Example

The following statement supplies a missing middle initial and changes the job for employee 000200.

```
UPDATE YEMP
  SET MIDINIT = 'H', JOB = 'FIELDREP'
  WHERE EMPNO = '000200';
```

The following statement gives everyone in department D11 a raise of 400.00. The statement can update several rows.

```
UPDATE YEMP
  SET SALARY = SALARY + 400.00
  WHERE WORKDEPT = 'D11';
```

The following statement sets the salary for employee 000190 to the average salary and sets the bonus to the minimum bonus for all employees.

```
UPDATE YEMP
  SET (SALARY, BONUS) =
  (SELECT AVG(SALARY), MIN(BONUS)
    FROM EMP)
   WHERE EMPNO = '000190';
```

Related reference

UPDATE (Db2 SQL)

Selecting values while updating data

When you update rows in a table, you can select the updated values from those rows at the same time.

Procedure

Specify the UPDATE statement in the FROM clause of the SELECT statement.

When you update one or more rows in a table, you can retrieve:

- The value of an automatically generated column such as a ROWID or identity column
- Any default values for columns
- All values for an updated row, without specifying individual column names

In most cases, you want to use the FINAL TABLE clause with SELECT FROM UPDATE statements. The FINAL TABLE consists of the rows of the table or view after the update occurs.

Examples

Example: SELECT FROM FINAL TABLE

Suppose that all clerks for a company are receiving 5 percent raises. You can use the following SELECT FROM UPDATE statement to increase the salary of each designer by 5 percent and to retrieve the total increase in salary for the company.

```
SELECT SUM(SALARY) INTO :salary FROM FINAL TABLE
  (UPDATE EMP SET SALARY = SALARY * 1.05
   WHERE JOB = 'DESIGNER');
```

Example: retrieving data row-by-row from updated data

To retrieve row-by-row output of updated data, use a cursor with a SELECT FROM UPDATE statement. For example, suppose that all designers for a company are receiving a 30 percent increase in their bonus. You can use the following SELECT FROM UPDATE statement to increase the bonus of each clerk by 30 percent and to retrieve the bonus for each clerk.

```
DECLARE CS1 CURSOR FOR
  SELECT LASTNAME, BONUS FROM FINAL TABLE
   (UPDATE EMP SET BONUS = BONUS * 1.3
    WHERE JOB = 'CLERK');
FETCH CS1 INTO :lastname, :bonus;
```

Example: INCLUDE a new column in the result table but not the target table

You can use the INCLUDE clause to introduce a new column to the result table but not add the column to the target table. For example, suppose that sales representatives received a 20 percent increase in their commission. You need to update the commission (COMM) of sales representatives (SALESREP) in the EMP table and that you need to retrieve the old commission and the new commission for each sales representative. You can use the following SELECT FROM UPDATE statement to perform the update and to retrieve the required data.

```
DECLARE CS2 CURSOR FOR
SELECT LASTNAME, COMM, old_comm FROM FINAL TABLE
  (UPDATE EMP INCLUDE(old_comm DECIMAL (7,2))
   SET COMM = COMM * 1.2, old_comm = COMM
   WHERE JOB = 'SALESREP');
```

Related reference

table-reference (Db2 SQL)

UPDATE (Db2 SQL)

Updating thousands of rows

When you update large volumes of data, consider certain recommended actions to increase concurrency.

About this task

Question: Are there any special techniques for updating large volumes of data?

Answer: Yes. When updating large volumes of data using a cursor, you can minimize the amount of time that you hold locks on the data by declaring the cursor with the HOLD option and by issuing commits frequently.

Deleting data from tables

You can delete data from a table by deleting one or more rows from the table, by deleting all rows from the table, or by dropping columns from the table.

Procedure

To delete one or more rows in a table:

- Use the DELETE statement with a WHERE clause to specify a search condition.

  The DELETE statement removes zero or more rows of a table, depending on how many rows satisfy the search condition that you specify in the WHERE clause.

  You can use DELETE with a WHERE clause to remove only selected rows from a declared temporary table, but not from a created temporary table.

  The following DELETE statement deletes each row in the YEMP table that has an employee number '000060'.

  ```
  DELETE FROM YEMP
    WHERE EMPNO = '000060';
  ```

  When this statement executes, Db2 deletes any row from the YEMP table that meets the search condition.

If Db2 finds an error while executing your DELETE statement, it stops deleting data and returns error codes in the SQLCODE and SQLSTATE variables or related fields in the SQLCA. The data in the table does not change.

If the DELETE is successful, SQLERRD(3) in the SQLCA contains the number of deleted rows. This number includes only the number of deleted rows in the table that is specified in the DELETE statement. Rows that are deleted (in other tables) according to the CASCADE rule are not included in SQLERRD(3).

To delete every row in a table:

- Use the DELETE statement without specifying a WHERE clause.

  With segmented table spaces, deleting all rows of a table is very fast.

  The following DELETE statement deletes every row in the YDEPT table:

  ```
  DELETE FROM YDEPT;
  ```

  If the statement executes, the table continues to exist (that is, you can insert rows into it), but it is empty. All existing views and authorizations on the table remain intact when using DELETE.

- Use the TRUNCATE statement.

  The TRUNCATE statement can provide the following advantages over a DELETE statement:

  – The TRUNCATE statement can ignore delete triggers
  – The TRUNCATE statement can perform an immediate commit
  – The TRUNCATE statement can keep storage allocated for the table

  The TRUNCATE statement does not, however, reset the count for an automatically generated value for an identity column on the table. If 14872 was the next identity column value to be generated before a TRUNCATE statement, 14872 would be the next value generated after the TRUNCATE statement.

  Suppose that you need to empty the data from an old inventory table, regardless of any existing delete triggers, and you need to make the space that is allocated for the table available for other uses. Use the following TRUNCATE statement.

  ```
  TRUNCATE INVENTORY_TABLE
     IGNORE DELETE TRIGGERS
     DROP STORAGE;
  ```

  Suppose that you need to empty the data from an old inventory table permanently, regardless of any existing delete triggers, and you need to preserve the space that is allocated for the table. You need the emptied data to be completely unavailable, so that a ROLLBACK statement cannot return the data. Use the following TRUNCATE statement.

  ```
  TRUNCATE INVENTORY_TABLE
     REUSE STORAGE
     IGNORE DELETE TRIGGERS
     IMMEDIATE;
  ```

- Use the DROP TABLE statement.

  DROP TABLE drops the specified table and all related views and authorizations, which can invalidate plans and packages.

To drop columns from a table:

- Use the ALTER TABLE statement with the DROP COLUMN clause.

  Because dropping a column from a table is a pending change to the definition of the table, the table space is placed in advisory REORG-pending status (AREOR). When the pending change is applied (by running the REORG utility with the SHRLEVEL CHANGE or REFERENCE options), the column is dropped from the table, and any dependent packages and statements in the dynamic statement cache are invalidated.

Related concepts

SQL communication area (SQLCA) (Db2 SQL)

Related tasks

Dropping tables

When you drop a table, you delete the data and the table definition. You also delete all synonyms, views, indexes, referential constraints, and check constraints that are associated with that table.

Related reference

DELETE (Db2 SQL)

DROP (Db2 SQL)

TRUNCATE (Db2 SQL)

ALTER TABLE (Db2 SQL)

Selecting values while deleting data

When you delete rows from a table, you can select the values from those rows at the same time.

Procedure

Specify the DELETE statement in the FROM clause of the SELECT statement.

When you delete one or more rows in a table, you can retrieve:

- Any default values for columns
- All values for a deleted row, without specifying individual column names
- Calculated values based on deleted rows

Example

Example: FROM OLD TABLE clause

When you use a SELECT FROM DELETE statement, you must use the FROM OLD TABLE clause to retrieve deleted values. The OLD TABLE consists of the rows of the table or view before the delete occurs. For example, suppose that a company is eliminating all operator positions and that the company wants to know how much salary money it will save by eliminating these positions. You can use the following SELECT FROM DELETE statement to delete operators from the EMP table and to retrieve the sum of operator salaries.

```
SELECT SUM(SALARY) INTO :salary FROM OLD TABLE
  (DELETE FROM EMP
   WHERE JOB = 'OPERATOR');
```

Example: retrieving row-by-row output of deleted data

To retrieve row-by-row output of deleted data, use a cursor with a SELECT FROM DELETE statement. For example, suppose that a company is eliminating all analyst positions and that the company wants to know how many years of experience each analyst had with the company. You can use the following SELECT FROM DELETE statement to delete analysts from the EMP table and to retrieve the experience of each analyst.

```
DECLARE CS1 CURSOR FOR
SELECT YEAR(CURRENT DATE - HIREDATE) FROM OLD TABLE
  (DELETE FROM EMP
   WHERE JOB = 'ANALYST');
FETCH CS1 INTO :years_of_service;
```

Example: retrieving calculated data based on deleted dable

If you need to retrieve calculated data, based on the data that you delete but not add that column to the target table. For example, suppose that you need to delete managers from the EMP table and that you need to retrieve the salary and the years of employment for each manager. You can use the following SELECT FROM DELETE statement to perform the delete operation and to retrieve the required data.

```
DECLARE CS2 CURSOR FOR
SELECT LASTNAME, SALARY, years_employed FROM OLD TABLE
  (DELETE FROM EMP INCLUDE(years_employed INTEGER)
   SET years_employed = YEAR(CURRENT DATE - HIREDATE)
   WHERE JOB = 'MANAGER');
```

Related reference

table-reference (Db2 SQL)

DELETE (Db2 SQL)

Accessing data in tables from application programs

Your program can use a number of different techniques to read data from any Db2 tables for which you have read access. The simplest technique is to use basic SQL SELECT statements. However, you should choose the technique that works best for your situation and performs well.

About this task

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- IBM Data Server Manager (Introduction to Db2 for z/OS)
- Db2 Tools for z/OS

Related concepts

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Interpreting data access by using EXPLAIN (Db2 Performance)

Related tasks

Writing efficient SQL queries (Db2 Performance)

Determining which tables you have access to

You can ask Db2 to list the tables that a specific authorization ID has access to.

About this task

The contents of the Db2 catalog tables can be a useful reference tool when you begin to develop an SQL statement or an application program.

The catalog table, SYSIBM.SYSTABAUTH, lists table privileges that are granted to authorization IDs. To display the tables that you have authority to access (by privileges granted either to your authorization ID or to PUBLIC), you can execute an SQL statement similar to the one shown in the following example. To do this, you must have the SELECT privilege on SYSIBM.SYSTABAUTH.

Procedure

Issue a SELECT statement similar to the following example. To do this, you must have the SELECT privilege on SYSIBM.SYSTABAUTH.

```
SELECT DISTINCT TCREATOR, TTNAME
  FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*') AND GRANTEETYPE = ' ';
```

In this query, the predicate `GRANTEETYPE = ' '` selects authorization IDs.

Exception: If your Db2 subsystem uses an exit routine for access control authorization, you cannot rely on catalog queries to tell you the tables that you can access. When such an exit routine is installed, both RACF and Db2 control table access.

Related reference

SYSTABAUTH catalog table (Db2 SQL)

Explicit table and view privileges (Managing Security)

Displaying information about the columns for a given table

You can ask Db2 to list the columns in a particular table and certain information about those columns.

About this task

The catalog table, SYSIBM.SYSCOLUMNS, describes every column of every table.

Procedure

Query the SYSIBM.SYSCOLUMNS catalog table.

Examples

Example

Suppose that you want to display information about table DSN8B10.DEPT. If you have the SELECT privilege on SYSIBM.SYSCOLUMNS, you can use the following statement:

```
SELECT NAME, COLTYPE, SCALE, LENGTH
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME = 'DEPT'
  AND TBCREATOR = 'DSN8B10';
```

Example

If you display column information about a table that includes LOB or ROWID columns, the LENGTH field for those columns contains the number of bytes that those column occupy in the base table. The LENGTH field does not contain the length of the LOB or ROWID data.

To determine the maximum length of data for a LOB or ROWID column, include the LENGTH2 column in your query:

```
SELECT NAME, COLTYPE, LENGTH, LENGTH2
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME = 'EMP_PHOTO_RESUME'
  AND TBCREATOR = 'DSN8B10';
```

Related reference

SYSCOLUMNS catalog table (Db2 SQL)

Retrieving data by using the SELECT statement

The simplest way to retrieve data is to use the SQL statement SELECT to specify a result table. You can specify the columns and rows that you want to retrieve.

Before you begin

Consider developing your own SQL statements similar to the examples in this section, and then run them dynamically using SPUFI. For a tutorial see Lesson 1.1: Querying data interactively (Introduction to Db2 for z/OS).

You can also use the command line processor, or Db2 Query Management Facility (QMF).

Procedure

Issue a SELECT statement.

Examples

Example 1: Selecting all columns with SELECT *

You do not need to know the column names to select Db2 data. Use an asterisk (*) in the SELECT clause to indicate that you want to retrieve all columns of each selected row of the named table. Implicitly hidden columns, such as ROWID columns and XML document ID columns, are not included in the result of the SELECT * statement. To view the values of these columns, you must specify the column name.

The following SQL statement selects all columns from the department table:

```
SELECT *
  FROM DSN8B10.DEPT;
```

The result table looks similar to the following output:

```
DEPTNO  DEPTNAME                       MGRNO      ADMRDEPT  LOCATION
======  =============================  ======     ========  ========
A00     SPIFFY COMPUTER SERVICES DIV.  000010     A00       --------
B01     PLANNING                       000020     A00       --------
C01     INFORMATION CENTER             000030     A00       --------
D01     DEVELOPMENT CENTER             ------     A00       --------
D11     MANUFACTURING CENTER           000060     D01       --------
D21     ADMINISTRATION SYSTEMS         000070     D01       --------
E01     SUPPORT SERVICES               000050     A00       --------
E11     OPERATIONS                     000090     E01       --------
E21     SOFTWARE SUPPORT               000100     E01       --------
F22     BRANCH OFFICE F2               ------     E01       --------
G22     BRANCH OFFICE G2               ------     E01       --------
H22     BRANCH OFFICE H2               ------     E01       --------
I22     BRANCH OFFICE I2               ------     E01       --------
J22     BRANCH OFFICE J2               ------     E01       --------
```

Because the example does not specify a WHERE clause, the statement retrieves data from all rows.

The dashes for MGRNO and LOCATION in the result table indicate null values.

SELECT * is recommended mostly for use with dynamic SQL and view definitions. You can use SELECT * in static SQL, but doing so is not recommended because of possible host variable compatibility and performance implications. Suppose that you add a column to the table to which SELECT * refers. If you have not defined a receiving host variable for that column, an error might occur, or the data from the added column might not be retrieved.

If you list the column names in a static SELECT statement instead of using an asterisk, you can avoid problems that might occur with SELECT *. You can also see the relationship between the receiving host variables and the columns in the result table.

Example 2: selecting specific columns with SELECT *column-name*

Select the column or columns you want to retrieve by naming each column. With a single SELECT statement, you can select data from one column or as many as 750 columns. All columns appear in the order you specify, not in their order in the table.

For example, the following SQL statement retrieves only the MGRNO and DEPTNO columns from the department table:

```
SELECT MGRNO, DEPTNO
  FROM DSN8B10.DEPT;
```

The result table looks similar to the following output:

```
MGRNO     DEPTNO
======    ======
000010    A00
000020    B01
000030    C01
------    D01
000050    E01
000060    D11
000070    D21
000090    E11
```

```
000100    E21
------    F22
------    G22
------    H22
------    I22
------    J22
```

Example 3: Selecting data from implicity hidden columns

To SELECT data from implicitly hidden columns, such as ROWID and XML document ID, look up the column names in SYSIBM.SYSCOLUMNS and specify these names in the SELECT list. For example, suppose that you create and populate the following table:

```
CREATE TABLE MEMBERS (MEMBERID INTEGER,
                      BIO          XML,
                      REPORT      XML,
                      RECOMMENDATIONS     XML);
```

Db2 generates one additional implicitly hidden XML document ID column. To retrieve data in all columns, including the generated XML document ID column, first look up the name of the generated column in SYSIBM.SYSCOLUMNS. Suppose the name is DB2_GENERATED_DOCID_FOR_XML. Then, specify the following statement:

```
SELECT DB2_GENERATED_DOCID_FOR_XML, MEMBERID, BIO,
       REPORT, RECOMMENDATIONS FROM MEMBERS
```

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Remote servers and distributed data
Distributed data is data that resides on a database management system (DBMS) other than your local system. Your local DBMS is the one on which you bind your application plan. All other DBMSs are remote.

Predicates (Db2 SQL)

Related reference

select-statement (Db2 SQL)

Specifying search conditions with a WHERE clause

You can use a WHERE clause to select the rows that meet certain conditions. A WHERE clause specifies a search condition. A *search condition* consists of one or more predicates. A *predicate* specifies a test that you want Db2 to apply to each table row.

About this task

A WHERE clause specifies a search condition. A *search condition* consists of one or more predicates. A *predicate* specifies a test that you want Db2 to apply to each table row.

Procedure

Specify a WHERE clause with one or more predicates.

Db2 evaluates a predicate for each row as true, false, or unknown. Results are unknown only if an operand is null.

If a search condition contains a column of a distinct type, the value to which that column is compared must be of the same distinct type, or you must cast the value to the distinct type.

The following table lists the type of comparison, the comparison operators, and an example of each type of comparison that you can use in a predicate in a WHERE clause.

Table 66. Comparison operators used in conditions

Type of comparison	Comparison operator	Example
Equal to	=	DEPTNO = 'X01'
Not equal to	<>	DEPTNO <> 'X01'
Less than	<	AVG(SALARY) < 30000
Less than or equal to	<=	AGE <= 25
Not less than	>=	AGE >= 21
Greater than	>	SALARY > 2000
Greater than or equal to	>=	SALARY >= 5000
Not greater than	<=	SALARY <= 5000
Equal to null	IS NULL	PHONENO IS NULL
Not equal to another value or one value is equal to null	IS DISTINCT FROM	PHONENO IS DISTINCT FROM :PHONEHV
Similar to another value	LIKE	NAME LIKE ' or STATUS LIKE 'N_'
At least one of two conditions	OR	HIREDATE < '1965-01-01' OR SALARY < 16000
Both of two conditions	AND	HIREDATE < '1965-01-01' AND SALARY < 16000
Between two values	BETWEEN	SALARY BETWEEN 20000 AND 40000
Equals a value in a set	IN (X, Y, Z)	DEPTNO IN ('B01', 'C01', 'D01')
Note: SALARY BETWEEN 20000 AND 40000 is equivalent to SALARY >= 20000 AND SALARY <= 40000.		

You can also search for rows that do not satisfy one of the preceding conditions by using the NOT keyword before the specified condition.

You can search for rows that do not satisfy the IS DISTINCT FROM predicate by using either of the following predicates:

- *value 1* IS NOT DISTINCT FROM *value 2*
- NOT(*value 1* IS DISTINCT FROM *value 2*)

Both of these forms of the predicate create an expression for which one value is equal to another value or both values are equal to null.

Related concepts

Subqueries
When you need to narrow your search condition based on information in an interim table, you can use a subquery. For example, you might want to find all employee numbers in one table that also exist for a given project in a second table.

Predicates (Db2 SQL)

Related tasks

Coding SQL statements to avoid unnecessary processing (Db2 Performance)

Related reference

where-clause (Db2 SQL)

Handling null values

A null value indicates the absence of a column value in a row. A null value is an unknown value; it is not the same as zero or all blanks.

About this task

Null values can be used as a condition in the WHERE and HAVING clauses. For example, a WHERE clause can specify a column that, for some rows, contains a null value. A basic comparison predicate using a column that contains null values does not select a row that has a null value for the column. This is because a null value is not less than, equal to, or greater than the value specified in the condition. The IS NULL predicate is used to check for null values.

Examples

Example 1: Selecting rows that contain null in a column

To select the values for all rows that contain a null value for the manager number, you can issue the following statement:

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT
  FROM DSN8B10.DEPT
  WHERE MGRNO IS NULL
```

The following table shows the result.

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00
F22	BRANCH OFFICE F2	E01
G22	BRANCH OFFICE G2	E01
H22	BRANCH OFFICE H2	E01
I22	BRANCH OFFICE I2	E01
J22	BRANCH OFFICE J2	E01

Example 2: Selecting rows that do not contain a null value

To get the rows that do not have a null value for the manager number, you can change the WHERE clause in the previous example like this:

```
WHERE MGRNO IS NOT NULL
```

Example 3: Comparing values that contain the NULL value

Another predicate that is useful for comparing values that can contain the NULL value is the DISTINCT predicate. Comparing two columns using a normal equal comparison (COL1 = COL2) will be true if both columns contain an equal non-null value. If both columns are null, the result will be false because null is never equal to any other value, not even another null value. Using the DISTINCT predicate, null values are considered equal. So COL1 is NOT DISTINCT from COL2 will be true if both columns contain an equal non-null value and also when both columns are the null value.

For example, suppose that you want to select information from two tables that contain null values. The first table T1 has a column C1 with the following values.

C1
2
1
null

The second table has column C2 with the following values.

C2
2
null

Assume that you issue the following SELECT statement:

```
SELECT *
    FROM T1, T2
    WHERE C1 IS DISTINCT FROM C2
```

The result follows.

C1	C2
1	2
1	-
2	-
-	2

Related concepts

Nulls (Db2 SQL)

Null values in table columns (Introduction to Db2 for z/OS)

How to check for null values

Before you retrieve a column value, you might first want to determine if the column value is null.

Applications frequently need to check two values to see if they are equal or not equal. You can use a basic predicate to do an equal or not equal comparison. An equal comparison or a not equal comparison can return true, false, or unknown. The normal rule in SQL, except for the DISTINCT predicate, is that one null value is never equal to another null value. If either or both operands of a basic predicate are the null value, the result is unknown.

Depending on your application, you might want to include or exclude rows that have a NULL value in a column. You can use the NULL predicate to do that.

MY_EMP is a table that has a row with the last name and phone number for each of the employees in a company. In this company, no employees share a phone number, but some employees might not have a phone number. The LASTNAME column contains the last name of each employee. The PHONENO column contains the phone number for each employee. If an employee does not have a phone, the PHONENO column value is NULL. The table might look like this:

LASTNAME	PHONENO
HAAS	-------
THOMPSON	3476
KWAN	4738
GEYER	6789
STERN	6423

Suppose that you want to know the last name of the employee who has no phone number. Using a query like this one does not work, because if the PHONENO column value is NULL, the WHERE clause compares a NULL column value to a null host variable value. The result of that comparison is unknown.

```
MOVE -1 TO PHONENO-IND.
EXEC SQL
 SELECT LASTNAME
  INTO :LASTNAME-HV
  FROM MY_EMP
  WHERE PHONENO = :PHONENO-HV :PHONENO-IND
END-EXEC.
```

To find the employee with a NULL value for the phone number, you need to use a NULL predicate:

```
EXEC SQL
 SELECT LASTNAME
  INTO :LASTNAME-HV
  FROM MY_EMP
  WHERE PHONENO IS NULL
END-EXEC.
```

The SELECT statement returns a LASTNAME value of 'HAAS'.

Now suppose that you want to select the last name of an employee whose phone number matches a certain value or whose phone number is NULL. To do that, you need to code two search conditions: one to handle the case where the phone number is not NULL, and another to handle the case where the phone number is NULL. The SELECT statement might look like this:

```
EXEC SQL
  SELECT LASTNAME
  INTO :LASTNAME-HV
  FROM MY_EMP
  WHERE (PHONENO IS NOT NULL AND :PHONENO-HV :PHONENO-IND IS NOT NULL
         AND PHONENO = :PHONENO-HV )  -- Search condition for non-NULL
                                      -- phone number
         OR
        (PHONENO IS NULL AND :PHONENO-HV :PHONENO-IND IS NULL)
                                      -- Search condition for NULL
                                      -- phone number
END-EXEC.
```

If you set :PHONENO-HV to '3476' and :PHONENO-IND to 0, the SELECT statement returns 'THOMPSON' because the search condition for a non-NULL phone number is used. If you set :PHONENO-HV to any value, and set :PHONENO-IND to -1, the SELECT statement returns 'HAAS' because the search condition for a NULL phone number is used.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Related reference

DISTINCT predicate (Db2 SQL)

NULL predicate (Db2 SQL)

Selecting derived columns

In a SELECT statement, you can select columns that are not actual columns in a table. Instead, you can specify "columns" that are derived from a constant, an expression, or a function.

Example: SELECT with an expression

This SQL statement generates a result table in which the second column is a derived column that is generated by adding the values of the SALARY, BONUS, and COMM columns.

```
SELECT EMPNO, (SALARY + BONUS + COMM)
   FROM DSN8B10.EMP;
```

Derived columns in a result table, such as (SALARY + BONUS + COMM), do not have names. You can use the AS clause to give a name to an unnamed column of the result table. For information about using the AS clause, see "Naming result columns" on page 365.

What to do next

To order the rows in a result table by the values in a derived column, specify a name for the column by using the AS clause, and specify that name in the ORDER BY clause. For information about using the ORDER BY clause, see "Ordering the result table rows" on page 366.

Selecting XML data

You can select all XML data that is stored in a particular column or only a subset of data from an XML column.

Procedure

- You can select all XML data that is stored in a particular column by specifying SELECT *column name* or SELECT *, just as you would for columns of any other data type.
- Alternatively, you can select only a subset of data from an XML column by using an XPath expression in a SELECT statement. XPath expressions identify specific nodes in an XML document.

  To select a subset of data in an XML column, specify the XMLQUERY function in your SELECT statement with the following parameters:

  – An XPath expression that is embedded in a character string constant. Specify an XPath expression that identifies which XML data to return.

  – Any additional values to pass to the XPath expression, including the XML column name. Specify these values after the PASSING keyword.

Example

Suppose that you store purchase orders as XML documents in the POrder column in the PurchaseOrders table. You need to find in each purchase order the items whose product name is equal to a name in the Product table. You can use the following statement to find these values:

```
SELECT XMLQUERY('//item[productName = $n]'
  PASSING PO.POrder,
  P.name AS "n")
FROM PurchaseOrders PO, Product P;
```

This statement returns the item elements in the POrder column that satisfy the criteria in the XPath expression.

Related concepts

Overview of XQuery (Db2 Programming for XML)

Related reference

XMLQUERY (Db2 SQL)

Formatting the result table

An SQL statement returns data in a table called a result table. You can specify certain attributes of the result table, such as the column names, how the rows are ordered, and whether the rows are numbered.

Result tables

The data that is retrieved by an SQL statement is always in the form of a table, which is called a *result table*. Like the tables from which you retrieve the data, a result table has rows and columns. A program fetches this data one row at a time.

Example result table: Assume that you issue the following SELECT statement, which retrieves the last name, first name, and phone number of employees in department D11 from the sample employee table:

```
SELECT LASTNAME, FIRSTNME, PHONENO
  FROM DSN8B10.EMP
  WHERE WORKDEPT = 'D11'
  ORDER BY LASTNAME;
```

The result table looks similar to the following output:

```
LASTNAME             FIRSTNME           PHONENO
================     ==============     ==========
ADAMSON              BRUCE              4510
BROWN                DAVID              4501
JOHN                 REBA               0672
JONES                WILLIAM            0942
LUTZ                 JENNIFER           0672
PIANKA               ELIZABETH          3782
SCOUTTEN             MARILYN            1682
STERN                IRVING             6432
WALKER               JAMES              2986
YAMAMOTO             KIYOSHI            2890
YOSHIMURA            MASATOSHI          2890
```

Excluding duplicate rows from the result table of a query

You can ask Db2 to exclude multiple identical rows from a query result table. For example, a query might return multiple rows for each employee when one row per employee is sufficient for your program.

Procedure

Specify the DISTINCT keyword in the query.

The DISTINCT keyword excludes duplicate rows from your query result table, so that each row contains unique data.

Restriction: You cannot use the DISTINCT keyword with LOB columns or XML columns.

Example

The following SELECT statement lists unique department numbers for administrative departments:

```
SELECT DISTINCT ADMRDEPT
  FROM DSN8B10.DEPT;
```

The result table looks similar to the following output:

```
ADMRDEPT
========
A00
D01
E01
```

Related tasks

Coding SQL statements to avoid unnecessary processing (Db2 Performance)

Related reference

select-clause (Db2 SQL)

Naming result columns

You can provide your own names for the result table columns for a SELECT statement. This capability is particularly useful for a column that is derived from an expression or a function.

Procedure

Use the AS clause to name result columns in a SELECT statement.

Examples

The following examples show different ways to use the AS clause.

Example: SELECT with AS CLAUSE

The following example of the SELECT statement gives the expression SALARY+BONUS+COMM the name TOTAL_SAL.

```
SELECT SALARY+BONUS+COMM AS TOTAL_SAL
   FROM DSN8B10.EMP
   ORDER BY TOTAL_SAL;
```

Example: CREATE VIEW with AS clause

You can specify result column names in the select-clause of a CREATE VIEW statement. You do not need to supply the column list of CREATE VIEW, because the AS keyword names the derived column. The columns in the view EMP_SAL are EMPNO and TOTAL_SAL.

```
CREATE VIEW EMP_SAL AS
  SELECT EMPNO,SALARY+BONUS+COMM AS TOTAL_SAL
     FROM DSN8B10.EMP;
```

Example: set operator with AS clause

You can use the AS clause with set operators, such as UNION. In this example, the AS clause is used to give the same name to corresponding columns of tables in a UNION. The third result column from the union of the two tables has the name TOTAL_VALUE, even though it contains data that is derived from columns with different names:

```
SELECT 'On hand' AS STATUS, PARTNO, QOH * COST AS TOTAL_VALUE
   FROM PART_ON_HAND
UNION ALL
SELECT 'Ordered' AS STATUS, PARTNO, QORDER * COST AS TOTAL_VALUE
   FROM ORDER_PART
ORDER BY PARTNO, TOTAL_VALUE;
```

The column STATUS and the derived column TOTAL_VALUE have the same name in the first and second result tables. They are combined in the union of the two result tables, which is similar to the following partial output:

```
STATUS    PARTNO   TOTAL_VALUE
=======   ======   ===========
On hand   00557    345.60
Ordered   00557    150.50
.
.
.
```

Example: GROUP BY derived column

You can use the AS clause in a FROM clause to assign a name to a derived column that you want to refer to in a GROUP BY clause. This SQL statement names HIREYEAR in the nested table expression, which lets you use the name of that result column in the GROUP BY clause:

```
SELECT HIREYEAR, AVG(SALARY)
   FROM (SELECT YEAR(HIREDATE) AS HIREYEAR, SALARY
            FROM DSN8B10.EMP) AS NEWEMP
   GROUP BY HIREYEAR;
```

You cannot use GROUP BY with a name that is defined with an AS clause for the derived column YEAR(HIREDATE) in the outer SELECT, because that name does not exist when the GROUP BY runs.

However, you can use GROUP BY with a name that is defined with an AS clause in the nested table expression, because the nested table expression runs before the GROUP BY that references the name.

Related tasks

Combining result tables from multiple SELECT statements
When you combine the results of multiple SELECT statements, you can choose what to include in the result table. You can include all rows, only rows that are in the result table of both SELECT statements, or only rows that are unique to the result table of the first SELECT statement.

Defining a view
A *view* is a named specification of a result table. Use views to control which users have access to certain data or to simplify writing SQL statements.

Summarizing group values
You can group rows in the result table by the values of one or more columns or by the results of an expression. You can then apply aggregate functions to each group.

Related reference

select-clause (Db2 SQL)

Ordering the result table rows

If you want to guarantee that the rows in your result table are ordered in a particular way, you must specify the order in the SELECT statement. Otherwise, Db2 can return the rows in any order.

About this task

Using ORDER BY is the only way to guarantee that your rows are ordered as you want them.

Procedure

To retrieve rows in a specific order, use the ORDER BY clause

Examples

Example: Specifying the sort key in the ORDER BY clause

The order of the selected rows depends on the sort keys that you identify in the ORDER BY clause. A *sort key* can be a column name, an integer that represents the number of a column in the result table, or an expression. Db2 orders the rows by the first sort key, followed by the second sort key, and so on.

You can list the rows in ascending or descending order. Null values appear last in an ascending sort and first in a descending sort.

Db2 sorts strings in the collating sequence associated with the encoding scheme of the table. Db2 sorts numbers algebraically and sorts datetime values chronologically.

Restriction: You cannot use the ORDER BY clause with LOB or XML columns.

Example: ORDER BY clause with a column name as the sort key

Retrieve the employee numbers, last names, and hire dates of employees in department A00 in ascending order of hire dates:

```
SELECT EMPNO, LASTNAME, HIREDATE
   FROM DSN8B10.EMP
   WHERE WORKDEPT = 'A00'
   ORDER BY HIREDATE ASC;
```

The result table looks similar to the following output:

```
EMPNO    LASTNAME       HIREDATE
=====    =========      ==========
000110   LUCCHESI       1958-05-16
000120   O'CONNELL      1963-12-05
000010   HAAS           1965-01-01
```

```
200010   HEMMINGER      1965-01-01
200120   ORLANDO        1972-05-05
```

Example: ORDER BY clause with an expression as the sort key

The following subselect retrieves the employee numbers, salaries, commissions, and total compensation (salary plus commission) for employees with a total compensation greater than 40000. Order the results by total compensation:

```
SELECT EMPNO, SALARY, COMM, SALARY+COMM AS "TOTAL COMP"
   FROM DSN8B10.EMP
   WHERE SALARY+COMM > 40000
   ORDER BY SALARY+COMM;
```

The intermediate result table looks similar to the following output:

```
EMPNO   SALARY    COMM      TOTAL COMP
======  ========  =======   ==========
000030  38250.00  3060.00   41310.00
000050  40175.00  3214.00   43389.00
000020  41250.00  3300.00   44550.00
000110  46500.00  3720.00   50220.00
200010  46500.00  4220.00   50720.00
000010  52750.00  4220.00   56970.00
```

Referencing derived columns in the ORDER BY clause

If you use the AS clause to name an unnamed column in a SELECT statement, you can use that name in the ORDER BY clause. The following SQL statement orders the selected information by total salary:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL
   FROM DSN8B10.EMP
   ORDER BY TOTAL_SAL;
```

Related reference

fullselect (Db2 SQL)

Numbering the rows in a result table

Db2 does not number the rows in the result table for a query unless you explicitly request that the rows be numbered.

About this task

To number the rows in a result table, include the ROW_NUMBER specification in your query. If you want to ensure that the rows are in a particular order, include an ORDER BY clause after the OVER keyword. Otherwise, the rows are numbered in an arbitrary order.

Example

Suppose that you want a list of employees and salaries from department D11 in the sample EMP table. You can return a numbered list that is ordered by last name by submitting the following query:

```
SELECT ROW_NUMBER() OVER (ORDER BY LASTNAME) AS NUMBER,
WORKDEPT, LASTNAME, SALARY
FROM DSN8910.EMP
WHERE WORKDEPT='D11'
```

This query returns the following result:

```
---------+---------+---------+---------+---------+---------+-----
NUMBER                   WORKDEPT  LASTNAME            SALARY
---------+---------+---------+---------+---------+---------+-----
                      1  D11       ADAMSON           25280.00
                      2  D11       BROWN             27740.00
                      3  D11       JOHN              29840.00
                      4  D11       JONES             18270.00
                      5  D11       LUTZ              29840.00
                      6  D11       PIANKA            22250.00
                      7  D11       SCOUTTEN          21340.00
                      8  D11       STERN             32250.00
```

```
                    9  D11       WALKER              20450.00
                   10  D11       YAMAMOTO            24680.00
                   11  D11       YOSHIMURA           24680.00
```

Related reference

OLAP specification (Db2 SQL)

Ranking the rows

You can request that Db2 calculate the ordinal rank of each row in the result set based on a particular column. For example, you can rank finishing times for a marathon to determine the first, second, and third place finishers.

Procedure

To rank rows, use one of the following ranking specifications in an SQL statement:

- Use RANK to return a rank number for each row value.

  Use this specification if you want rank numbers to be skipped when duplicate row values exist.

  For example, suppose the top five finishers in a marathon have the following times:

  - 2:31:57
  - 2:34:52
  - 2:34:52
  - 2:37:26
  - 2:38:01

  When you use the RANK specification, Db2 returns the following rank numbers:

  Table 67. Example of values returned when you specify RANK

Value	Rank number
2:31:57	1
2:34:52	2
2:34:52	2
2:37:26	4
2:38:01	5

- Use DENSE_RANK to return a rank number for each row value.

  Use this specification if you do not want rank numbers to be skipped when duplicate row values exist.

  For example, when you specify DENSE_RANK with the same times that are listed in the description of RANK, Db2 returns the following rank numbers:

  Table 68. Example of values returned when you specify RANK

Value	Rank number
2:31:57	1
2:34:52	2
2:34:52	2
2:37:26	3
2:38:01	4

Examples

Suppose that you had the following values in the DATA column of table T1:

```
DATA
-------
100
 35
 23
  8
  8
  6
```

Example: RANK

Suppose that you use the following RANK specification:

```
SELECT DATA,
  RANK() OVER (ORDER BY DATA DESC) AS RANK_DATA
  FROM T1
  ORDER BY RANK_DATA;
```

Db2 returns the following ranked data:

```
DATA        RANK_DATA
-------------------
 100                1
  35                2
  23                3
   8                4
   8                4
   6                6
```

Example: DENSE RANK

Suppose that you use the following DENSE_RANK specification on the same data:

```
SELECT DATA,
  DENSE_RANK() OVER (ORDER BY DATA DESC) AS RANK_DATA
  FROM T1
  ORDER BY RANK_DATA;
```

Db2 returns the following ranked data:

```
DATA        RANK_DATA
-------------------
 100                1
  36                2
  23                3
   8                4
   8                4
   6                5
```

In the example with the RANK specification, two equal values are both ranked as 4. The next rank number is 6. Number 5 is skipped.

In the example with the DENSE_RANK option, those two equal values are also ranked as 4. However, the next rank number is 5. With DENSE_RANK, no gaps exist in the sequential rank numbering.

Related reference

OLAP specification (Db2 SQL)

Combining result tables from multiple SELECT statements

When you combine the results of multiple SELECT statements, you can choose what to include in the result table. You can include all rows, only rows that are in the result table of both SELECT statements, or only rows that are unique to the result table of the first SELECT statement.

About this task

Assume that you want to combine the results of two SELECT statements that return the following result tables:

Example: R1 result table

```
COL1     COL2
a        a
a        b
a        c
```

Example: R2 result table

```
COL1     COL2
a        b
a        c
a        d
```

You can use the *set operators* to combine two or more SELECT statements to form a single result table:

UNION

UNION returns all of the values from the result table of each SELECT statement. If you want all duplicate rows to be repeated in the result table, specify UNION ALL. If you want redundant duplicate rows to be eliminated from the result table, specify UNION or UNION DISTINCT.

For example, the following example is the result of specifying UNION for R1 and R2.

```
COL1     COL2
a        a
a        b
a        c
a        d
```

EXCEPT

Returns all rows from the first result table (R1) that are not also in the second result table (R2). If you want all duplicate rows from R1 to be contained in the result table, specify EXCEPT ALL. If you want redundant duplicate rows in R1 to be eliminated from the result table, specify EXCEPT or EXCEPT DISTINCT.

The result of the EXCEPT operation depends on the which SELECT statement is included before the EXCEPT keyword in the SQL statement. For example, if the SELECT statement that returns the R1 result table is listed first, the result is a single row:

```
COL1     COL2
a        a
```

If the SELECT statement that returns the R2 result table is listed first, the final result is a different row:

```
COL1     COL2
a        d
```

INTERSECT

Returns rows that are in the result table of both SELECT statements. If you want all duplicate rows to be contained in the result table, specify INTERSECT ALL. If you want redundant duplicate rows to be eliminated from the result table, specify INTERSECT or INTERSECT DISTINCT.

For example, the following example is the result of specifying UNION for R1 and R2.

```
COL1     COL2
a        b
a        c
```

When you specify one of the set operators, Db2 processes each SELECT statement to form an interim result table, and then combines the interim result table of each statement. If the *n*th column of the first result table (R1) and the *n*th column of the second result table (R2) have the same result column name, the *n*th column of the result table has that same result column name. If the *n*th column of R1 and the nth column of R2 do not have the same names, the result column is unnamed.

Procedure

- To combine two or more SELECT statements to form a single result table, use the set operators: UNION, EXCEPT or INTERSECT.
  For example, assume that you have the following tables to manage stock at two book stores.

Table 69. STOCKA

ISBN	TITLE	AUTHOR	NOBEL PRIZE
8778997709	For Whom the Bell Tolls	Hemmingway	N
4599877699	The Good Earth	Buck	Y
9228736278	A Tale of Two Cities	Dickens	N
1002387872	Beloved	Morrison	Y
4599877699	The Good Earth	Buck	Y
0087873532	The Labyrinth of Solitude	Paz	Y

Table 70. STOCKB

ISBN	TITLE	AUTHOR	NOBEL PRIZE
6689038367	The Grapes of Wrath	Steinbeck	Y
2909788445	The Silent Cry	Oe	Y
1182983745	Light in August	Faulkner	Y
9228736278	A Tale of Two Cities	Dickens	N
1002387872	Beloved	Morrison	Y

Example: UNION clause

Suppose that you want a list of books whose authors have won the Nobel Prize and that are in stock at either store. The following SQL statement returns these books in order by author name without redundant duplicate rows:

```
SELECT TITLE, AUTHOR
  FROM STOCKA
  WHERE NOBELPRIZE = 'Y'
UNION
SELECT TITLE, AUTHOR
  FROM STOCKB
  WHERE NOBELPRIZE = 'Y'
 ORDER BY AUTHOR
```

This statement returns the following final result table:

Table 71. Result of UNION	
TITLE	**AUTHOR**
The Good Earth	Buck
Light in August	Faulkner
Beloved	Morrison
The Silent Cry	Oe
The Labyrinth of Solitude	Paz
The Grapes of Wrath	Steinbeck

Example: EXCEPT clause

Suppose that you want a list of books that are only in STOCKA. The following SQL statement returns the book names that are in STOCKA only without any redundant duplicate rows:

```
SELECT TITLE
  FROM STOCKA
EXCEPT
SELECT TITLE
  FROM STOCKB
  ORDER BY TITLE;
```

This statement returns the following result table:

Table 72. Result of EXCEPT
TITLE
For Whom the Bell Tolls
The Good Earth
The Labyrinth of Solitude

Example: INTERSECT clause

Suppose that you want a list of books that are in both STOCKA and in STOCKB. The following statement returns a list of all books from both of these tables with redundant duplicate rows are removed.

```
SELECT TITLE
  FROM STOCKA
INTERSECT
SELECT TITLE
  FROM STOCKB
  ORDER BY TITLE;
```

This statement returns the following result table:

Table 73. Result of INTERSECT
TITLE
A Tale of Two Cities
Beloved

- To keep all duplicate rows when combining result tables, specify the ALL keyword with the set operator clause.

  The following examples use the STOCKA and STOCK B tables from the previous step.

Example: UNION ALL

The following SQL statement returns a list of books that won Nobel prizes and are in stock at either store, with duplicates included.

```
SELECT TITLE, AUTHOR
  FROM STOCKA
  WHERE NOBELPRIZE = 'Y'
UNION ALL
SELECT TITLE, AUTHOR
  FROM STOCKB
  WHERE NOBELPRIZE = 'Y'
 ORDER BY AUTHOR
```

This statement returns the following result table:

Table 74. Result of UNION ALL

TITLE	AUTHOR
The Good Earth	Buck
The Good Earth	Buck
Light in August	Faulkner
Beloved	Morrison
Beloved	Morrison
The Silent Cry	Oe
The Labyrinth of Solitude	Paz
The Grapes of Wrath	Steinbeck

Example: EXCEPT ALL

Suppose that you want a list of books that are only in STOCKA. The following SQL statement returns the book names that are in STOCKA only with all duplicate rows:

```
SELECT TITLE
  FROM STOCKA
EXCEPT ALL
SELECT TITLE
  FROM STOCKB
  ORDER BY TITLE;
```

This statement returns the following result table:

Table 75. Result of EXCEPT ALL

TITLE
For Whom the Bell Tolls
The Good Earth
The Good Earth
The Labyrinth of Solitude

Example: INTERSECT ALL

Suppose that you want a list of books that are in both STOCKA and in STOCKB, including any duplicate matches. The following statement returns a list of titles that are in both stocks, including duplicate matches. In this case, one match exists for "A Tale of Two Cities" and one match exists for "Beloved."

```
SELECT TITLE
  FROM STOCKA
INTERSECT ALL
```

```
SELECT TITLE
  FROM STOCKB
  ORDER BY TITLE;
```

This statement returns the following result table:

Table 76. Result of INTERSECT ALL

TITLE
A Tale of Two Cities
Beloved

- To eliminate redundant duplicate rows when combining result tables, specify one of the following keywords:
  - UNION or UNION DISTINCT
  - EXCEPT or EXCEPT DISTINCT
  - INTERSECT or INTERSECT DISTINCT
- To order the entire result table, specify the ORDER BY clause at the end.

Related tasks

Ordering the result table rows
If you want to guarantee that the rows in your result table are ordered in a particular way, you must specify the order in the SELECT statement. Otherwise, Db2 can return the rows in any order.

Related reference

fullselect (Db2 SQL)

Summarizing group values

You can group rows in the result table by the values of one or more columns or by the results of an expression. You can then apply aggregate functions to each group.

Procedure

Use the GROUP BY clause.

When it is used, the GROUP BY clause follows the FROM clause and any WHERE clause, and it precedes the ORDER BY clause.

Except for the columns that are named in the GROUP BY clause, the SELECT statement must specify any other selected columns as an operand of one of the aggregate functions.

If a column that you specify in the GROUP BY clause contains null values, Db2 considers those null values to be equal. Thus, all nulls form a single group.

Examples

Example: GROUP BY clause using one column

The following SQL statement lists, for each department, the lowest and highest education level within that department:

```
SELECT WORKDEPT, MIN(EDLEVEL), MAX(EDLEVEL)
   FROM DSN8B10.EMP
   GROUP BY WORKDEPT;
```

Example: GROUP BY clause using more than one column

You can group the rows by the values of more than one column. For example, The following statement finds the average salary for men and women in departments A00 and C01:

```
SELECT WORKDEPT, SEX, AVG(SALARY) AS AVG_SALARY
   FROM DSN8B10.EMP
```

```
   WHERE WORKDEPT IN ('A00', 'C01')
   GROUP BY WORKDEPT, SEX;
```

The result table looks similar to the following output:

```
WORKDEPT   SEX     AVG_SALARY
========   ===     ==============
A00        F       49625.00000000
A00        M       35000.00000000
C01        F       29722.50000000
```

Db2 groups the rows first by department number and then (within each department) by sex before it derives the average SALARY value for each group.

Example: GROUP BY clause using a expression

You can also group the rows by the results of an expression. For example, the following statement groups departments by their leading characters, and lists the lowest and highest education level for each group:

```
SELECT SUBSTR(WORKDEPT,1,1), MIN(EDLEVEL), MAX(EDLEVEL)
   FROM DSN8B10.EMP
   GROUP BY SUBSTR(WORKDEPT,1,1);
```

Related reference

group-by-clause (Db2 SQL)

Filtering groups

If you group rows in the result table, you can also specify a search condition that each retrieved group must satisfy. The search condition tests properties of each group rather than properties of individual rows in the group.

Procedure

Use the HAVING clause to specify a search condition.

The HAVING clause acts like a WHERE clause for groups, and it contains the same kind of search conditions that you specify in a WHERE clause.

Example

Example: HAVING clause

The following SQL statement includes a HAVING clause that specifies a search condition for groups of work departments in the employee table:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY
   FROM DSN8B10.EMP
   GROUP BY WORKDEPT
   HAVING COUNT(*) > 1
   ORDER BY WORKDEPT;
```

The result table looks similar to the following output:

```
WORKDEPT   AVG_SALARY
========   ==============
A00        40850.00000000
C01        29722.50000000
D11        25147.27272727
D21        25668.57142857
E11        21020.00000000
E21        24086.66666666
```

Compare the preceding example with the second example shown in "Summarizing group values" on page 374. The clause, `HAVING COUNT(*) > 1`, ensures that only departments with more than one member are displayed. In this case, departments B01 and E01 do not display because the HAVING clause tests a property of the group.

Example: HAVING clause used with a GROUP BY clause

Use the HAVING clause to retrieve the average salary and minimum education level of women in each department for which all female employees have an education level greater than or equal to 16. Assuming that you want results from only departments A00 and D11, the following SQL statement tests the group property, MIN(EDLEVEL):

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY,
   MIN(EDLEVEL) AS MIN_EDLEVEL
   FROM DSN8B10.EMP
   WHERE SEX = 'F' AND WORKDEPT IN ('A00', 'D11')
   GROUP BY WORKDEPT
   HAVING MIN(EDLEVEL) >= 16;
```

The result table looks similar to the following output:

```
WORKDEPT   AVG_SALARY       MIN_EDLEVEL
========   ==============   ===========
A00        49625.00000000   18
D11        25817.50000000   17
```

When you specify both GROUP BY and HAVING, the HAVING clause must follow the GROUP BY clause. A function in a HAVING clause can include DISTINCT if you have not used DISTINCT anywhere else in the same SELECT statement. You can also connect multiple predicates in a HAVING clause with AND or OR, and you can use NOT for any predicate of a search condition.

Related reference

where-clause (Db2 SQL)

having-clause (Db2 SQL)

Finding rows that were changed within a specified period of time

You can filter rows based on the time that they were updated. For example, you might want to find all rows in a particular table that have been changed in the last 7 days.

Procedure

Specify the ROW CHANGE TIMESTAMP expression in the predicate of your SQL statement.

Recommendation: Ensure that the table has a ROW CHANGE TIMESTAMP column that was defined prior to the time period that you want to query. This column ensures that Db2 returns only those rows that were updated in the given time period.

If the table does not have a ROW CHANGE TIMESTAMP column, Db2 returns all rows on each page that has had any changes within the given time period. In this case, your result set can contain rows that have not been updated in the give time period, if other rows on that page have been updated or inserted.

Examples

Example

Suppose that the TAB table has a ROW CHANGE TIMESTAMP column and that you want to return all of the records that have changed in the last 30 days. The following query returns all of those rows.

```
SELECT * FROM TAB
WHERE ROW CHANGE TIMESTAMP FOR TAB <= CURRENT TIMESTAMP AND
ROW CHANGE TIMESTAMP FOR TAB >= CURRENT TIMESTAMP - 30 days;
```

Example

Suppose that you want to return all of the records that have changed since 9:00 AM January 1, 2004. The following query returns all of those rows.

```
SELECT * FROM TAB
WHERE ROW CHANGE TIMESTAMP FOR TAB >= '2004-01-01-09.00.00';
```

Related reference
ROW CHANGE expression (Db2 SQL)
CREATE TABLE (Db2 SQL)
where-clause (Db2 SQL)

Joining data from more than one table

Sometimes the information that you want to see is not in a single table. To form a row of the result table, you might want to retrieve some column values from one table and some column values from another table.

About this task

You can use a SELECT statement to retrieve and join column values from two or more tables into a single row.

A join operation typically matches a row of one table with a row of another on the basis of a join condition. Db2 supports the following types of joins: inner join, left outer join, right outer join, and full outer join. You can specify joins in the FROM clause of a query.

Examples

Nested table expressions and user-defined table functions in joins
An operand of a join can be more complex than the name of a single table. You can specify one of the following items as a join operand:

nested table expression
A fullselect that is enclosed in parentheses and followed by a correlation name. The correlation name lets you refer to the result of that expression.

Using a nested table expression in a join can be helpful when you want to create a temporary table to use in a join. You can specify the nested table expression as either the right or left operand of a join, depending on which unmatched rows you want included.

user-defined table function
A user-defined function that returns a table.

Using a nested table expression in a join can be helpful when you want to perform some operation on the values in a table before you join them to another table.

Example: Using correlated references
In the following SELECT statement, the correlation name that is used for the nested table expression is CHEAP_PARTS. You can use this correlation name to refer to the columns that are returned by the expression. In this case, those correlated references are CHEAP_PARTS.PROD# and CHEAP_PARTS.PRODUCT.

```
SELECT CHEAP_PARTS.PROD#, CHEAP_PARTS.PRODUCT
   FROM (SELECT PROD#, PRODUCT
            FROM PRODUCTS
            WHERE PRICE < 10) AS CHEAP_PARTS;
```

The result table looks similar to the following output:

```
PROD#       PRODUCT
=====       ===========
505         SCREWDRIVER
30          RELAY
```

The correlated references are valid because they do not occur in the table expression where CHEAP_PARTS is defined. The correlated references are from a table specification at a higher level in the hierarchy of subqueries.

Example: Using a nested table expression as the right operand of a join

The following query contains a fullselect (in bold) as the right operand of a left outer join with the PROJECTS table. The correlation name is TEMP. In this case the unmatched rows from the PROJECTS table are included, but the unmatched rows from the nested table expression are not.

```
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
FROM PROJECTS LEFT JOIN
      (SELECT PART,
          COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
          PRODUCTS.PRODUCT
       FROM PARTS FULL OUTER JOIN PRODUCTS
             ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
   ON PROJECTS.PROD# = PRODNUM;
```

Example: Using a nested table expression as the left operand of a join

The following query contains a fullselect as the left operand of a left outer join with the PRODUCTS table. The correlation name is PARTX. In this case the unmatched rows from the nested table expression are included, but the unmatched rows from the PRODUCTS table are not.

```
SELECT PART, SUPPLIER, PRODNUM, PRODUCT
  FROM (SELECT PART, PROD# AS PRODNUM, SUPPLIER
           FROM PARTS
           WHERE PROD# < '200') AS PARTX
        LEFT OUTER JOIN PRODUCTS
           ON PRODNUM = PROD#;
```

The result table looks similar to the following output:

```
PART        SUPPLIER         PRODNUM     PRODUCT
=======     ============     =======     ==========
WIRE        ACWF             10          GENERATOR
MAGNETS     BATEMAN          10          GENERATOR
OIL         WESTERN_CHEM     160         ----------
```

Because PROD# is a character field, Db2 does a character comparison to determine the set of rows in the result. Therefore, because the characters '30' are greater than '200', the row in which PROD# is equal to '30' does not appear in the result.

Example: Using a table function as an operand of a join

Suppose that CVTPRICE is a table function that converts the prices in the PRODUCTS table to the currency that you specify and returns the PRODUCTS table with the prices in those units. You can obtain a table of parts, suppliers, and product prices with the prices in your choice of currency by executing a query similar to the following query:

```
SELECT PART, SUPPLIER, PARTS.PROD#, Z.PRODUCT, Z.PRICE
   FROM PARTS, TABLE(CVTPRICE(:CURRENCY)) AS Z
   WHERE PARTS.PROD# = Z.PROD#;
```

Correlated references in table specifications in joins

Use correlation names to refer to the results of a nested table expression. After you specify the correlation name for an expression, any subsequent reference to this correlation name is called a *correlated reference*.

You can include correlated references in nested table expressions or as arguments to table functions. The basic rule that applies for both of these cases is that the correlated reference must be from a table specification at a higher level in the hierarchy of subqueries. You can also use a correlated reference and the table specification to which it refers in the same FROM clause if the table specification appears to the left of the correlated reference and the correlated reference is in one of the following clauses:

- A nested table expression that is preceded by the keyword TABLE
- The argument of a table function

For more information about correlated references, see "Correlation names in references" on page 395.

A table function or a table expression that contains correlated references to other tables in the same FROM clause cannot participate in a full outer join or a right outer join. The following examples illustrate valid uses of correlated references in table specifications.

In this example, the correlated reference T.C2 is valid because the table specification, to which it refers, T, is to its left.

```
SELECT T.C1, Z.C5
  FROM T, TABLE(TF3(T.C2)) AS Z
  WHERE T.C3 = Z.C4;
```

If you specify the join in the opposite order, with T following TABLE(TF3(T.C2), T.C2 is invalid.

In this example, the correlated reference D.DEPTNO is valid because the nested table expression within which it appears is preceded by TABLE, and the table specification D appears to the left of the nested table expression in the FROM clause.

```
SELECT D.DEPTNO, D.DEPTNAME,
  EMPINFO.AVGSAL, EMPINFO.EMPCOUNT
  FROM DEPT D,
    TABLE(SELECT AVG(E.SALARY) AS AVGSAL,
            COUNT(*) AS EMPCOUNT
            FROM EMP E
            WHERE E.WORKDEPT=D.DEPTNO) AS EMPINFO;
```

If you remove the keyword TABLE, D.DEPTNO is invalid.

Related reference

from-clause (Db2 SQL)

Joining more than two tables

Joins are not limited to two tables. You can join more than two tables in a single SQL statement.

Procedure

Specify join conditions that include columns from all of the relevant tables.

Example

Example: Joining three tables

Suppose that you want a result table that shows employees who have projects that they are responsible for, their projects, and their department names. You need to join three tables to get all the information. You can use the following SELECT statement:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM DSN8B10.EMP, DSN8B10.PROJ, DSN8B10.DEPT
       WHERE EMPNO = RESPEMP
      AND WORKDEPT = DSN8B10.DEPT.DEPTNO;
```

The result table looks similar to the following output:

```
EMPNO   LASTNAME         DEPTNAME                      PROJNO
======  =========        ===========================   ======
000010  HAAS             SPIFFY COMPUTER SERVICE DIV   AD3100
000010  HAAS             SPIFFY COMPUTER SERVICE DIV   MA2100
000020  THOMPSON         PLANNING                      PL2100
000030  KWAN             INFORMATION CENTER            IF1000
000030  KWAN             INFORMATION CENTER            IF2000
000050  GEYER            SUPPORT SERVICES              OP1000
000050  GEYER            SUPPORT SERVICES              OP2000
000060  STERN            MANUFACTURING SYSTEMS         MA2110
000070  PULASKI          ADMINISTRATION SYSTEMS        AD3110
000090  HENDERSON        OPERATIONS                    OP1010
000100  SPENSER          SOFTWARE SUPPORT              OP2010
000150  ADAMSON          MANUFACTURING SYSTEMS         MA2112
000160  PIANKA           MANUFACTURING SYSTEMS         MA2113
000220  LUTZ             MANUFACTURING SYSTEMS         MA2111
000230  JEFFERSON        ADMINISTRATION SYSTEMS        AD3111
000250  SMITH            ADMINISTRATION SYSTEMS        AD3112
```

```
000270  PEREZ           ADMINISTRATION SYSTEMS         AD3113
000320  MEHTA           SOFTWARE SUPPORT               OP2011
000330  LEE             SOFTWARE SUPPORT               OP2012
000340  GOUNOT          SOFTWARE SUPPORT               OP2013
```

Db2 determines the intermediate and final results of the previous query by performing the following logical steps:

1. Join the employee and project tables on the employee number, dropping the rows with no matching employee number in the project table.
2. Join the intermediate result table with the department table on matching department numbers.
3. Process the select list in the final result table, leaving only four columns.

Example: Joining more than two tables by using more than one join type

When joining more than two tables, you do not have to use the same join type for every join.

To join tables by using more than one join type, specify the join types in the FROM clause.

Suppose that you want a result table that shows the following items:

- Employees whose last name begins with 'S' or a letter that comes after 'S' in the alphabet
- The department names for the these employees
- Any projects that these employees are responsible for

You can use the following SELECT statement:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM DSN8B10.EMP INNER JOIN DSN8B10.DEPT
       ON WORKDEPT = DSN8B10.DEPT.DEPTNO
    LEFT OUTER JOIN DSN8B10.PROJ
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S';
```

The result table looks like similar to the following output:

```
EMPNO   LASTNAME         DEPTNAME                  PROJNO
======  =========        ======================    ======
000020  THOMPSON         PLANNING                  PL2100
000060  STERN            MANUFACTURING SYSTEMS     MA2110
000100  SPENSER          SOFTWARE SUPPORT          OP2010
000170  YOSHIMURA        MANUFACTURING SYSTEMS     ------
000180  SCOUTTEN         MANUFACTURING SYSTEMS     ------
000190  WALKER           MANUFACTURING SYSTEMS     ------
000250  SMITH            ADMINISTRATION SYSTEMS    AD3112
000280  SCHNEIDER        OPERATIONS                ------
000300  SMITH            OPERATIONS                ------
000310  SETRIGHT         OPERATIONS                ------
200170  YAMAMOTO         MANUFACTURING SYSTEMS     ------
200280  SCHWARTZ         OPERATIONS                ------
200310  SPRINGER         OPERATIONS                ------
200330  WONG             SOFTWARE SUPPORT          ------
```

Db2

determines the intermediate and final results of the previous query by performing the following logical steps:

1. Join the employee and department tables on matching department numbers, dropping the rows where the last name begins with a letter before 'S in the alphabet'.
2. Join the intermediate result table with the project table on the employee number, keeping the rows for which no matching employee number exists in the project table.
3. Process the select list in the final result table, leaving only four columns.

Related reference

from-clause (Db2 SQL)

Inner joins

An *inner join* is a method of combining two tables that discards rows of either table that do not match any row of the other table. The matching is based on the join condition.

To request an inner join, execute a SELECT statement in which you specify the tables that you want to join in the FROM clause, and specify a WHERE clause or an ON clause to indicate the join condition. The join condition can be any simple or compound search condition that does not contain a subquery reference.

In the simplest type of inner join, the join condition is *column1=column2*.

Example

You can join the PARTS and PRODUCTS tables in sample data from joins on the PROD# column to get a table of parts with their suppliers and the products that use the parts.

To do this, you can use either one of the following SELECT statements:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
  WHERE PARTS.PROD# = PRODUCTS.PROD#;
```

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result table looks like the following output:

```
PART      SUPPLIER        PROD#   PRODUCT
=======   ============    =====   =========
WIRE      ACWF            10      GENERATOR
MAGNETS   BATEMAN         10      GENERATOR
PLASTIC   PLASTIK_CORP    30      RELAY
BLADES    ACE_STEEL       205     SAW
```

Three things about this example:

- A part in the parts table (OIL) has product (#160), which is not in the products table. A product (SCREWDRIVER, #505) has no parts listed in the parts table. Neither OIL nor SCREWDRIVER appears in the result of the join.

  In contrast, an *outer join* includes rows in which the values in the joined columns do not match.
- You can explicitly specify that this join is an inner join (not an outer join). Use INNER JOIN in the FROM clause instead of the comma, and use ON to specify the join condition (rather than WHERE) when you explicitly join tables in the FROM clause.
- If you do not specify a WHERE clause in the first form of the query, the result table contains all possible combinations of rows for the tables that are identified in the FROM clause. You can obtain the same result by specifying a join condition that is always true in the second form of the query, as in the following statement:

  ```
  SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
    FROM PARTS INNER JOIN PRODUCTS
      ON 1=1;
  ```

  Regardless of whether you omit the WHERE clause or specify a join condition that is always true, the number of rows in the result table is the product of the number of rows in each table.

You can specify more complicated join conditions to obtain different sets of results. For example, to eliminate the suppliers that begin with the letter **A** from the table of parts, suppliers, product numbers, and products, write a query like the following query:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
    AND SUPPLIER NOT LIKE 'A%';
```

The result of the query is all rows that do not have a supplier that begins with A. The result table looks like the following output:

```
PART       SUPPLIER       PROD#     PRODUCT
=======    ============   =====     ==========
MAGNETS    BATEMAN        10        GENERATOR
PLASTIC    PLASTIK_CORP   30        RELAY
```

Example of joining a table to itself by using an inner join

Joining a table to itself is useful to show relationships between rows. The following example returns a list of major projects from the PROJ table and the projects that are part of those major projects.

In this example, **A** indicates the first instance of table DSN8B10.PROJ, and **B** indicates the second instance of this table. The join condition is such that the value in column PROJNO in table DSN8B10.PROJ A must be equal to a value in column MAJPROJ in table DSN8B10.PROJ B.

The following SQL statement joins table DSN8B10.PROJ to itself and returns the number and name of each major project followed by the number and name of the project that is part of it:

```
SELECT A.PROJNO, A.PROJNAME, B.PROJNO, B.PROJNAME
  FROM DSN8B10.PROJ A, DSN8B10.PROJ B
  WHERE A.PROJNO = B.MAJPROJ;
```

The result table looks similar to the following output:

```
PROJNO  PROJNAME                 PROJNO   PROJNAME
======  =======================  =======  ========================
AD3100  ADMIN SERVICES           AD3110   GENERAL AD SYSTEMS
AD3110  GENERAL AD SYSTEMS       AD3111   PAYROLL PROGRAMMING
AD3110  GENERAL AD SYSTEMS       AD3112   PERSONNEL PROGRAMMG
⋮
OP2010  SYSTEMS SUPPORT          OP2013   DB/DC SUPPORT
```

In this example, the comma in the FROM clause implicitly specifies an inner join, and it acts the same as if the INNER JOIN keywords had been used. When you use the comma for an inner join, you must specify the join condition on the WHERE clause. When you use the INNER JOIN keywords, you must specify the join condition on the ON clause.

Related reference

Sample data for joins
You can use the sample PARTS table and the PRODUCTS table to practice various types of joins.

from-clause (Db2 SQL)

Outer joins

An *outer join* is a method of combining two or more tables so that the result includes unmatched rows of one of the tables, or of both tables. The matching is based on the join condition.

Db2 supports three types of outer joins:

full outer join
Includes unmatched rows from both tables. If any column of the result table does not have a value, that column has the null value in the result table.

left outer join
Includes rows from the table that is specified before LEFT OUTER JOIN that have no matching values in the table that is specified after LEFT OUTER JOIN.

right outer join
Includes rows from the table that is specified after RIGHT OUTER JOIN that have no matching values in the table that is specified before RIGHT OUTER JOIN.

The following table illustrates how the PARTS and PRODUCTS tables in "Sample data for joins" on page 386 can be combined using the three outer join functions.

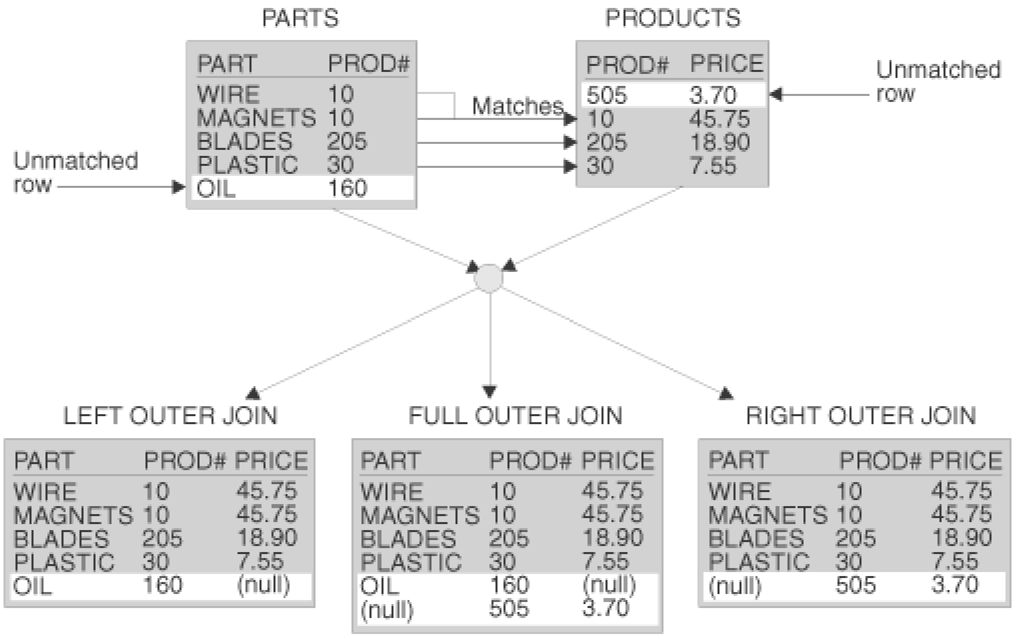


Figure 18. Three outer joins from the PARTS and PRODUCTS tables

The result table contains data that is joined from all of the tables, for rows that satisfy the search conditions.

The result columns of a join have names if the outermost SELECT list refers to base columns. However, if you use a function (such as COALESCE or VALUE) to build a column of the result, that column does not have a name unless you use the AS clause in the SELECT list.

Full outer join

An *full outer join* is a method of combining tables so that the result includes unmatched rows of both tables.

If you are joining two tables and want the result set to include unmatched rows from both tables, use a FULL OUTER JOIN clause. The matching is based on the join condition. If any column of the result table does not have a value, that column has the null value in the result table.

The join condition for a full outer join must be a simple search condition that compares two columns or an invocation of a cast function that has a column name as its argument.

Example: The following query performs a full outer join of the PARTS and PRODUCTS tables in "Sample data for joins" on page 386:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result table from the query looks similar to the following output:

```
PART         SUPPLIER       PROD#     PRODUCT
=======      ============   =====     ==========
WIRE         ACWF           10        GENERATOR
MAGNETS      BATEMAN        10        GENERATOR
PLASTIC      PLASTIK_CORP   30        RELAY
BLADES       ACE_STEEL      205       SAW
OIL          WESTERN_CHEM   160       -----------
-------      ------------   ---       SCREWDRIVER
```

Example of using COALESCE or VALUE: COALESCE is the keyword that is specified by the SQL standard as a synonym for the VALUE function. This function, by either name, can be particularly useful in full outer join operations because it returns the first non-null value from the pair of join columns.

The product number in the result of the example for “Full outer join” on page 383 is null for SCREWDRIVER, even though the PRODUCTS table contains a product number for SCREWDRIVER. If you select PRODUCTS.PROD# instead, PROD# is null for OIL. If you select both PRODUCTS.PROD# and PARTS.PROD#, the result contains two columns, both of which contain some null values. You can merge data from both columns into a single column, eliminating the null values, by using the COALESCE function.

With the same PARTS and PRODUCTS tables, the following example merges the non-null data from the PROD# columns:

```
SELECT PART, SUPPLIER,
   COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
    FROM PARTS FULL OUTER JOIN PRODUCTS
       ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result table looks similar to the following output:

```
PART        SUPPLIER       PRODNUM   PRODUCT
=======     ============   =======   ===========
WIRE        ACWF           10        GENERATOR
MAGNETS     BATEMAN        10        GENERATOR
PLASTIC     PLASTIK_CORP   30        RELAY
BLADES      ACE_STEEL      205       SAW
OIL         WESTERN_CHEM   160       -----------
-------     ------------   505       SCREWDRIVER
```

The AS clause (AS PRODNUM) provides a name for the result of the COALESCE function.

Left outer join

A *left outer join* is a method of combining tables. The result includes unmatched rows from only the table that is specified before the LEFT OUTER JOIN clause.

If you are joining two tables and want the result set to include unmatched rows from only one table, use a LEFT OUTER JOIN clause or a RIGHT OUTER JOIN clause. The matching is based on the join condition.

The clause LEFT OUTER JOIN includes rows from the table that is specified before LEFT OUTER JOIN that have no matching values in the table that is specified after LEFT OUTER JOIN.

As in an inner join, the join condition can be any simple or compound search condition that does not contain a subquery reference.

Example: The following example uses the tables in “Sample data for joins” on page 386. To include rows from the PARTS table that have no matching values in the PRODUCTS table, and to include prices that exceed 10.00, run the following query:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT, PRICE
  FROM PARTS LEFT OUTER JOIN PRODUCTS
    ON PARTS.PROD#=PRODUCTS.PROD#
       AND PRODUCTS.PRICE>10.00;
```

The result table looks similar to the following output:

```
PART        SUPPLIER       PROD#     PRODUCT        PRICE
=======     ============   =====     ==========     =====
WIRE        ACWF           10        GENERATOR      45.75
MAGNETS     BATEMAN        10        GENERATOR      45.75
PLASTIC     PLASTIK_CORP   30        -----------  -------
BLADES      ACE_STEEL      205       SAW            18.90
OIL         WESTERN_CHEM   160       -----------  -------
```

A row from the PRODUCTS table is in the result table only if its product number matches the product number of a row in the PARTS table and the price is greater than 10.00 for that row. Rows in which the PRICE value does not exceed 10.00 are included in the result of the join, but the PRICE value is set to null.

In this result table, the row for PROD# 30 has null values on the right two columns because the price of PROD# 30 is less than 10.00. PROD# 160 has null values on the right two columns because PROD# 160 does not match another product number.

Right outer join

A *right outer join* is a method of combining tables. The result includes unmatched rows from only the table that is specified after the RIGHT OUTER JOIN clause.

If you are joining two tables and want the result set to include unmatched rows from only one table, use a LEFT OUTER JOIN clause or a RIGHT OUTER JOIN clause. The matching is based on the join condition.

The clause RIGHT OUTER JOIN includes rows from the table that is specified after RIGHT OUTER JOIN that have no matching values in the table that is specified before RIGHT OUTER JOIN.

As in an inner join, the join condition can be any simple or compound search condition that does not contain a subquery reference.

Example: The following example uses the tables in "Sample data for joins" on page 386. To include rows from the PRODUCTS table that have no corresponding rows in the PARTS table, execute this query:

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT, PRICE
  FROM PARTS RIGHT OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
       AND PRODUCTS.PRICE>10.00;
```

The result table looks similar to the following output:

```
PART        SUPPLIER       PROD#     PRODUCT        PRICE
=======     ============   =====     ==========     =====
WIRE        ACWF            10       GENERATOR      45.75
MAGNETS     BATEMAN         10       GENERATOR      45.75
BLADES      ACE_STEEL      205       SAW            18.90
----------  ------------    30       RELAY           7.55
----------  ------------   505       SCREWDRIVER     3.70
```

A row from the PARTS table is in the result table only if its product number matches the product number of a row in the PRODUCTS table and the price is greater than 10.00 for that row.

Because the PRODUCTS table can have rows with nonmatching product numbers in the result table, and the PRICE column is in the PRODUCTS table, rows in which PRICE is less than or equal to 10.00 are included in the result. The PARTS columns contain null values for these rows in the result table.

SQL rules for statements that contain join operations

Typically, Db2 performs a join operation first, before it evaluates the other clauses of the SELECT statement.

SQL rules dictate that the result of a SELECT statement look as if the clauses had been evaluated in this order:

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT

A join operation is part of a FROM clause; therefore, for the purpose of predicting which rows will be returned from a SELECT statement that contains a join operation, assume that the join operation is performed first.

Example: Suppose that you want to obtain a list of part names, supplier names, product numbers, and product names from the PARTS and PRODUCTS tables. You want to include rows from either table where the PROD# value does not match a PROD# value in the other table, which means that you need to do a full outer join. You also want to exclude rows for product number 10. Consider the following SELECT statement:

```
SELECT PART, SUPPLIER,
  VALUE(PARTS.PROD#,PRODUCTS.PROD#) AS PRODNUM, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
```

```
    ON PARTS.PROD# = PRODUCTS.PROD#
    WHERE PARTS.PROD# <> '10' AND PRODUCTS.PROD# <> '10';
```

The following result is **not** what you wanted:

```
PART        SUPPLIER       PRODNUM   PRODUCT
=======     ============   =======   ===========
PLASTIC     PLASTIK_CORP   30        RELAY
BLADES      ACE_STEEL      205       SAW
```

Db2 performs the join operation first. The result of the join operation includes rows from one table that do not have corresponding rows from the other table. However, the WHERE clause then excludes the rows from both tables that have null values for the PROD# column.

The following statement is a correct SELECT statement to produce the list:

```
SELECT PART, SUPPLIER,
  VALUE(X.PROD#, Y.PROD#) AS PRODNUM, PRODUCT
  FROM
    (SELECT PART, SUPPLIER, PROD# FROM PARTS WHERE PROD# <> '10') X
    FULL OUTER JOIN
    (SELECT PROD#, PRODUCT FROM PRODUCTS WHERE PROD# <> '10') Y
    ON X.PROD# = Y.PROD#;
```

For this statement, Db2 applies the WHERE clause to each table separately. Db2 then performs the full outer join operation, which includes rows in one table that do not have a corresponding row in the other table. The final result includes rows with the null value for the PROD# column and looks similar to the following output:

```
PART        SUPPLIER       PRODNUM   PRODUCT
=======     ============   =======   ===========
OIL         WESTERN_CHEM   160       -----------
BLADES      ACE_STEEL      205       SAW
PLASTIC     PLASTIK_CORP   30        RELAY
-------     ------------   505       SCREWDRIVER
```

Sample data for joins

You can use the sample PARTS table and the PRODUCTS table to practice various types of joins.

The examples in these topics use the following two tables to show various types of joins:

```
      The PARTS table                        The PRODUCTS table
PART      PROD#   SUPPLIER              PROD#    PRODUCT        PRICE
=======   =====   ============          =====    ===========    =====
WIRE      10      ACWF                  505      SCREWDRIVER    3.70
OIL       160     WESTERN_CHEM          30       RELAY          7.55
MAGNETS   10      BATEMAN               205      SAW            18.90
PLASTIC   30      PLASTIK_CORP          10       GENERATOR      45.75
BLADES    205     ACE_STEEL
```

Optimizing retrieval for a small set of rows

When you need only a few of the thousands of rows that satisfy a query, you can tell Db2 to optimize its retrieval process to return only a specified number of rows.

About this task

Question: How can I tell Db2 that I want only a few of the thousands of rows that satisfy a query?

Answer: Use the optimize clause or the fetch clause of the SELECT statement.

Db2 usually optimizes queries to retrieve all rows that qualify. But sometimes you want to retrieve a few rows. For example, to retrieve the first row that is greater than or equal to a known value, code you SELECT statement like the following:

```
SELECT column list FROM table
WHERE key >= value
ORDER BY key ASC
```

Even with the ORDER BY clause, Db2 might fetch all the data first and sort it after the fetch, which could impact performance. Instead, you can write the query in one of the following ways:

```
SELECT * FROM table
WHERE key >= value
ORDER BY key ASC
OPTIMIZE FOR 1 ROW
```

```
SELECT * FROM table
WHERE key >= value
ORDER BY key ASC
FETCH FIRST n ROWS ONLY
```

Use OPTIMIZE FOR 1 ROW clause to influence the access path. OPTIMIZE FOR 1 ROW tells Db2 to select an access path that returns the first qualifying row quickly.

Use FETCH FIRST *n* ROWS ONLY clause to limit the number of rows in the result table to *n* rows. FETCH FIRST *n* ROWS ONLY has the following benefits:

- When you use FETCH statements to retrieve data from a result table, the fetch clause causes Db2 to retrieve only the number of rows that you need. This can have performance benefits, especially in distributed applications. If you try to execute a FETCH statement to retrieve the *n*+1st row, Db2 returns a +100 SQLCODE.
- When you use fetch clause in a SELECT INTO statement, you never retrieve more than one row. Using fetch clause in a SELECT INTO statement can prevent SQL errors that are caused by inadvertently selecting more than one value into a host variable.

When you specify the fetch clause but not the optimize clause, the optimize clause is implicit. When you specify FETCH FIRST *n* ROWS ONLY and OPTIMIZE FOR *m* ROWS, and *m* is less than *n*, Db2 optimizes the query for *m* rows. If *m* is greater than *n*, Db2 optimizes the query for *n* rows.

Related tasks

Fetching a limited number of rows (Db2 Performance)

Related reference

optimize-clause (Db2 SQL)

fetch-first-clause (Db2 SQL)

Creating recursive SQL by using common table expressions

Queries that use recursion are useful in applications like bill-of-materials applications, network planning applications, and reservation systems.

About this task

You can use common table expressions to create recursive SQL If a fullselect of a common table expression contains a reference to itself in a FROM clause, the common table expression is a *recursive common table expression*.

Recursive common table expressions must follow these rules:

- The first fullselect of the first union (the initialization fullselect) must not include a reference to the common table expression.
- Each fullselect that is part of the recursion cycle must:
  - Start with SELECT or SELECT ALL. SELECT DISTINCT is not allowed.
  - Include only one reference to the common table expression that is part of the recursion cycle in its FROM clause.
  - Not include aggregate functions, a GROUP BY clause, or a HAVING clause.
- The column names must be specified after the table name of the common table expression.
- The data type, length, and CCSID of each column from the common table expression must match the data type, length, and CCSID of each corresponding column in the iterative fullselect.

- If you use the UNION keyword, specify UNION ALL instead of UNION.
- You cannot specify INTERSECT or EXCEPT.
- Outer joins must not be part of any recursion cycle.
- A subquery must not be part of any recursion cycle.

Important: You should be careful to avoid an infinite loop when you use a recursive common table expression. Db2 issues a warning if one of the following items is **not** found in the iterative fullselect of a recursive common table expression:

- An integer column that increments by a constant
- A predicate in the WHERE clause in the form of *counter_column < constant* or *counter_column < :host variable*

See "Examples of recursive common table expressions" on page 147 for examples of bill-of-materials applications that use recursive common table expressions.

Updating data as it is retrieved from the database

As you retrieve rows, you can update them at the same time.

About this task

Question: How can I update rows of data as I retrieve them?

Answer: On the SELECT statement, use the FOR UPDATE clause without a column list, or the FOR UPDATE OF clause with a column list. For a more efficient program, specify a column list with only those columns that you intend to update. Then use the positioned UPDATE statement. The clause WHERE CURRENT OF identifies the cursor that points to the row you want to update.

Avoiding decimal arithmetic errors

When you request that Db2 perform a decimal operation, errors might occur if Db2 does not use the appropriate precision and scale.

About this task

For static SQL statements, the simplest way to avoid a division error is to override DEC31 rules by specifying the precompiler option DEC(15). In some cases you can avoid a division error by specifying D31.*s*, where *s* is a number between 1 and 9 and represents the minimum scale to be used for division operations. This specification reduces the probability of errors for statements that are embedded in the program.

If the dynamic SQL statements have bind, define, or invoke behavior and the value of the installation option for USE FOR DYNAMICRULES on panel DSNTIP4 is NO, you can use the precompiler option DEC(15), DEC15, or D15.*s* to override DEC31 rules, where *s* is a number between 1 and 9.

For a dynamic statement, or for a single static statement, use the scalar function DECIMAL to specify values of the precision and scale for a result that causes no errors.

Before you execute a dynamic statement, set the value of special register CURRENT PRECISION to DEC15 or D15.*s*, where *s* is a number between 1 and 9.

Even if you use DEC31 rules, multiplication operations can sometimes cause overflow because the precision of the product is greater than 31. To avoid overflow from multiplication of large numbers, use the MULTIPLY_ALT built-in function instead of the multiplication operator.

Precision for operations with decimal numbers

Db2 accepts two sets of rules for determining the precision and scale of the result of an operation with decimal numbers.

- DEC15 rules allow a maximum precision of 15 digits in the result of an operation. DEC15 rules are in effect when both operands have a precision of 15 or less, or unless the DEC31 rules apply.
- DEC31 rules allow a maximum precision of 31 digits in the result. DEC31 rules are in effect if any of the following conditions is true:
  - Either operand of the operation has a precision greater than 15 digits.
  - The operation is in a dynamic SQL statement, and any of the following conditions is true:
    - The current value of special register CURRENT PRECISION is DEC31 or D31.*s*, where *s* is a number between 1 and 9 and represents the minimum scale to be used for division operations.
    - The installation option for DECIMAL ARITHMETIC on panel DSNTIP4 is DEC31, 31, or D31.*s*, where *s* is a number between 1 and 9; the installation option for USE FOR DYNAMICRULES on panel DSNTIP4 is YES; and the value of CURRENT PRECISION has not been set by the application.
    - The SQL statement has bind, define, or invoke behavior; the statement is in an application that is precompiled with option DEC(31); the installation option for USE FOR DYNAMICRULES on panel DSNTIP4 is NO; and the value of CURRENT PRECISION has not been set by the application. See "Dynamic rules options for dynamic SQL statements" on page 867 for an explanation of bind, define, and invoke behavior.
  - The operation is in an embedded (static) SQL statement that you precompiled with the DEC(31), DEC31, or D31.*s* option, or with the default for that option when the installation option DECIMAL ARITHMETIC is DEC31 or 31. *s* is a number between 1 and 9 and represents the minimum scale to be used for division operations. See "Processing SQL statements" on page 829 for information about precompiling and for a list of all precompiler options.

Recommendation: To reduce the chance of overflow, or when dealing with a precision greater than 15 digits, choose DEC31 or D31.*s* , where*s* is a number between 1 and 9 and represents the minimum scale to be used for division operations.

Controlling how Db2 rounds decimal floating point numbers

You can specify a default rounding mode that Db2 is to use for all DECFLOAT values.

Procedure

Set the CURRENT DECFLOAT ROUNDING MODE special register.

Related reference

CURRENT DECFLOAT ROUNDING MODE (Db2 SQL)

SET CURRENT DECFLOAT ROUNDING MODE (Db2 SQL)

Implications of using SELECT *

Generally, you should use SELECT * only when you want to select all columns, except for hidden columns. Otherwise, specify the specific columns that you want to view.

Question: What are the implications of using SELECT * ?

Answer: Generally, you should select only the columns you need because Db2 is sensitive to the number of columns selected. Use SELECT * only when you are sure you want to select all columns, except hidden columns. (Hidden columns are not returned when you specify SELECT *.) One alternative to selecting all columns is to use views defined with only the necessary columns, and use SELECT * to access the views. Avoid SELECT * if all the selected columns participate in a sort operation (SELECT DISTINCT and SELECT...UNION, for example).

Subqueries

When you need to narrow your search condition based on information in an interim table, you can use a subquery. For example, you might want to find all employee numbers in one table that also exist for a given project in a second table.

Conceptual overview of subqueries

Suppose that you want a list of the employee numbers, names, and commissions of all employees who work on a particular project, whose project number is MA2111. The first part of the SELECT statement is easy to write:

```
SELECT EMPNO, LASTNAME, COMM
  FROM DSN8B10.EMP
  WHERE EMPNO
  ⋮
```

However, you cannot proceed because the DSN8B10.EMP table does not include project number data. You do not know which employees are working on project MA2111 without issuing another SELECT statement against the DSN8B10.EMPPROJACT table.

You can use a subquery to solve this problem. A *subquery* is a subselect or a fullselect in a WHERE clause. The SELECT statement that surrounds the subquery is called the *outer SELECT*.

```
SELECT EMPNO, LASTNAME, COMM
  FROM DSN8B10.EMP
  WHERE EMPNO IN
    (SELECT EMPNO
       FROM DSN8B10.EMPPROJACT
        WHERE PROJNO = 'MA2111');
```

To better understand the results of this SQL statement, imagine that Db2 goes through the following process:

1. Db2 evaluates the subquery to obtain a list of EMPNO values:

   ```
   (SELECT EMPNO
      FROM DSN8B10.EMPPROJACT
      WHERE PROJNO = 'MA2111');
   ```

   The result is in an interim result table, similar to the one in the following output:

   ```
   from EMPNO
        =====
        200
        200
        220
   ```

2. The interim result table then serves as a list in the search condition of the outer SELECT. Effectively, Db2 executes this statement:

   ```
   SELECT EMPNO, LASTNAME, COMM
     FROM DSN8B10.EMP
     WHERE EMPNO IN
     ('000200', '000220');
   ```

   As a consequence, the result table looks similar to the following output:

   ```
   EMPNO   LASTNAME  COMM
   ======  ========  ====
   000200  BROWN     2217
   000220  LUTZ      2387
   ```

Correlated and uncorrelated subqueries

Subqueries supply information that is needed to qualify a row (in a WHERE clause) or a group of rows (in a HAVING clause). The subquery produces a result table that is used to qualify the row or group of selected rows.

A subquery executes only once, if the subquery is the same for every row or group. This kind of subquery is *uncorrelated*, which means that it executes only once. For example, in the following statement, the content of the subquery is the same for every row of the table DSN8B10.EMP:

```
SELECT EMPNO, LASTNAME, COMM
  FROM DSN8B10.EMP
  WHERE EMPNO IN
    (SELECT EMPNO
       FROM DSN8B10.EMPPROJACT
        WHERE PROJNO = 'MA2111');
```

Subqueries that vary in content from row to row or group to group are *correlated* subqueries. For information about correlated subqueries, see "Correlated subqueries" on page 394.

Subqueries and predicates

A *predicate* is an element of a search condition that specifies a condition that is true, false, or unknown about a given row or group. A subquery, which is a SELECT statement within the WHERE or HAVING clause of another SQL statement, is always part of a predicate. The predicate is of the form:

```
operand operator (subquery)
```

A WHERE or HAVING clause can include predicates that contain subqueries. A predicate that contains a subquery, like any other search predicate, can be enclosed in parentheses, can be preceded by the keyword NOT, and can be linked to other predicates through the keywords AND and OR. For example, the WHERE clause of a query can look something like the following clause:

```
WHERE X IN (subquery1) AND (Y > SOME (subquery2) OR Z IS NULL)
```

Subqueries can also appear in the predicates of other subqueries. Such subqueries are nested subqueries at some level of nesting. For example, a subquery within a subquery within an outer SELECT has a nesting level of 2. Db2 allows nesting down to a level of 15, but few queries require a nesting level greater than 1.

The relationship of a subquery to its outer SELECT is the same as the relationship of a nested subquery to a subquery, and the same rules apply, except where otherwise noted.

The subquery result table

A subquery must produce a result table that has the same number of columns as the number of columns on the left side of the comparison operator. For example, both of the following SELECT statements are acceptable:

```
SELECT EMPNO, LASTNAME
  FROM DSN8B10.EMP
  WHERE SALARY =
  (SELECT AVG(SALARY)
    FROM DSN8B10.EMP);
```

```
SELECT EMPNO, LASTNAME
  FROM DSN8B10.EMP
  WHERE (SALARY, BONUS) IN
  (SELECT AVG(SALARY), AVG(BONUS)
    FROM DSN8B10.EMP);
```

Except for a subquery of a basic predicate, the result table can contain more than one row. For more information, see "Places where you can include a subquery" on page 392.

Related concepts

Subquery access (Db2 Performance)

Predicates (Db2 SQL)
Related tasks
Writing efficient subqueries (Db2 Performance)
Related reference
where-clause (Db2 SQL)
having-clause (Db2 SQL)

Places where you can include a subquery

You can specify a subquery in either a WHERE clause or a HAVING clause.

You can specify a subquery in either a WHERE or HAVING clause by using one of the following items:

Example: Basic predicate in a subquery

You can use a subquery immediately after any of the comparison operators. If you do, the subquery can return at most one value. Db2 compares that value with the value to the left of the comparison operator.

The following SQL statement returns the employee numbers, names, and salaries for employees whose education level is higher than the average company-wide education level.

```
SELECT EMPNO, LASTNAME, SALARY
  FROM DSN8B10.EMP
  WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
       FROM DSN8B10.EMP);
```

Example: Quantified predicate in a subquery: ALL, ANY, or SOME

You can use a subquery after a comparison operator, followed by the keyword ALL, ANY, or SOME. The number of columns and rows that the subquery can return for a quantified predicate depends on the type of quantified predicate:

- For = SOME, = ANY, or <> ALL, the subquery can return one or many rows and one or many columns. The number of columns in the result table must match the number of columns on the left side of the operator.
- For all other quantified predicates, the subquery can return one or many rows, but no more than one column.

See the information about quantified predicates, including what to do if a subquery that returns one or more null values gives you unexpected results.

Example: ALL predicate

Use ALL to indicate that the operands on the left side of the comparison must compare in the same way with **all** of the values that the subquery returns. For example, suppose that you use the greater-than comparison operator with ALL:

```
WHERE column > ALL (subquery)
```

To satisfy this WHERE clause, the column value must be greater than all of the values that the subquery returns. A subquery that returns an empty result table satisfies the predicate.

Now suppose that you use the <> operator with ALL in a WHERE clause like this:

```
WHERE (column1, column1, ... columnn) <> ALL (subquery)
```

To satisfy this WHERE clause, each column value must be unequal to all of the values in the corresponding column of the result table that the subquery returns. A subquery that returns an empty result table satisfies the predicate.

Example: ANY or SOME predicate

Use ANY or SOME to indicate that the values on the left side of the operator must compare in the indicated way to **at least one** of the values that the subquery returns. For example, suppose that you use the greater-than comparison operator with ANY:

```
WHERE expression > ANY (subquery)
```

To satisfy this WHERE clause, the value in the expression must be greater than at least one of the values (that is, greater than the lowest value) that the subquery returns. A subquery that returns an empty result table does not satisfy the predicate.

Now suppose that you use the = operator with SOME in a WHERE clause like this:

```
WHERE (column1, column1, ... columnn) = SOME (subquery)
```

To satisfy this WHERE clause, each column value must be equal to at least one of the values in the corresponding column of the result table that the subquery returns. A subquery that returns an empty result table does not satisfy the predicate.

Example: IN predicate in a subquery

You can use IN to say that the value or values on the left side of the IN operator must be among the values that are returned by the subquery. Using IN is equivalent to using = ANY or = SOME.

The following query returns the names of department managers:

```
SELECT EMPNO,LASTNAME
  FROM DSN8B10.EMP
    WHERE EMPNO IN
      (SELECT DISTINCT MGRNO
         FROM DSN8B10.DEPT);
```

EXISTS predicate in a subquery

When you use the keyword EXISTS, Db2 checks whether the subquery returns one or more rows. Returning one or more rows satisfies the condition; returning no rows does not satisfy the condition.

The search condition in the following query is satisfied if any project that is represented in the project table has an estimated start date that is later than 1 January 2005:

```
SELECT EMPNO,LASTNAME
  FROM DSN8B10.EMP
    WHERE EXISTS
      (SELECT *
         FROM DSN8B10.PROJ
         WHERE PRSTDATE > '2005-01-01');
```

The result of the subquery is always the same for every row that is examined for the outer SELECT. Therefore, either every row appears in the result of the outer SELECT or none appears. A correlated subquery is more powerful than the uncorrelated subquery that is used in this example because the result of a correlated subquery is evaluated for each row of the outer SELECT.

As shown in the example, you do not need to specify column names in the subquery of an EXISTS clause. Instead, you can code `SELECT *`. You can also use the EXISTS keyword with the NOT keyword in order to select rows when the data or condition that you specify does not exist; that is, you can code the following clause:

```
WHERE NOT EXISTS (SELECT ...);
```

Related tasks

Writing efficient subqueries (Db2 Performance)

Related reference

Quantified predicate (Db2 SQL)

having-clause (Db2 SQL)
where-clause (Db2 SQL)
EXISTS predicate (Db2 SQL)
IN predicate (Db2 SQL)

Correlated subqueries

A *correlated subquery* is a subquery that Db2 reevaluates when it examines a new row (in a WHERE clause) or a group of rows (in a HAVING clause) as it executes the outer SELECT statement.

In an uncorrelated subquery, Db2 executes the subquery once, substitutes the result of the subquery in the right side of the search condition, and evaluates the outer SELECT based on the value of the search condition.

User-defined functions in correlated subqueries

Use care when you invoke a user-defined function in a correlated subquery, and that user-defined function uses a scratchpad. Db2 does not refresh the scratchpad between invocations of the subquery. This can cause undesirable results because the scratchpad keeps values across the invocations of the subquery.

An example of a correlated subquery

Suppose that you want a list of all the employees whose education levels are higher than the average education levels in their respective departments. To get this information, Db2 must search the DSN8B10.EMP table. For each employee in the table, Db2 needs to compare the employee's education level to the average education level for that employee's department.

For this example, you need to use a correlated subquery, which differs from an uncorrelated subquery. An uncorrelated subquery compares the employee's education level to the average of the entire company, which requires looking at the entire table. A correlated subquery evaluates only the department that corresponds to the particular employee.

In the subquery, you tell Db2 to compute the average education level for the department number in the current row. The following query performs this action:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8B10.EMP X
  WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
       FROM DSN8B10.EMP
       WHERE WORKDEPT = X.WORKDEPT);
```

A correlated subquery looks like an uncorrelated one, except for the presence of one or more correlated references. In the example, the single correlated reference is the occurrence of X.WORKDEPT in the WHERE clause of the subselect. In this clause, the qualifier X is the correlation name that is defined in the FROM clause of the outer SELECT statement. X designates rows of the first instance of DSN8B10.EMP. At any time during the execution of the query, X designates the row of DSN8B10.EMP to which the WHERE clause is being applied.

Consider what happens when the subquery executes for a given row of DSN8B10.EMP. Before it executes, X.WORKDEPT receives the value of the WORKDEPT column for that row. Suppose, for example, that the row is for Christine Haas. Her work department is A00, which is the value of WORKDEPT for that row. Therefore, the following is the subquery that is executed for that row:

```
(SELECT AVG(EDLEVEL)
   FROM DSN8B10.EMP
   WHERE WORKDEPT = 'A00');
```

The subquery produces the average education level of Christine's department. The outer SELECT then compares this average to Christine's own education level. For some other row for which WORKDEPT has a different value, that value appears in the subquery in place of A00. For example, in the row for Michael

L Thompson, this value is B01, and the subquery for his row delivers the average education level for department B01.

The result table that is produced by the query is similar to the following output:

```
EMPNO    LASTNAME    WORKDEPT  EDLEVEL
======   =========   ========  =======
000010   HASS        A00       18
000030   KWAN        C01       20
000070   PULASKI     D21       16
000090   HENDERSON   E11       16
```

Related concepts

Correlated and non-correlated subqueries (Db2 Performance)

Related reference

having-clause (Db2 SQL)

where-clause (Db2 SQL)

Correlation names in references

A correlation name is a name that you specify for a table, view, nested table expression or table function. This name is valid only within the context in which it is defined. Use correlation names to avoid ambiguity, to establish correlated references, or to use shorter names for tables or views.

A correlated reference can appear in a subquery, in a nested table expression, or as an argument of a user-defined table function. For information about correlated references in nested table expressions and table functions, see "Joining data from more than one table" on page 377. In a subquery, the reference should be of the form X.C, where X is a correlation name and C is the name of a column in the table that X represents.

Any number of correlated references can appear in a subquery, with no restrictions on variety. For example, you can use one correlated reference in the outer SELECT, and another in a nested subquery.

When you use a correlated reference in a subquery, the correlation name can be defined in the outer SELECT or in any of the subqueries that contain the reference. Suppose, for example, that a query contains subqueries A, B, and C, and that A contains B and B contains C. The subquery C can use a correlation reference that is defined in B, A, or the outer SELECT.

You can define a correlation name for each table name in a FROM clause. Specify the correlation name after its table name. Leave one or more blanks between a table name and its correlation name. You can include the word AS between the table name and the correlation name to increase the readability of the SQL statement.

The following example demonstrates the use of a correlated reference in the search condition of a subquery:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8B10.EMP AS X
  WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
       FROM DSN8B10.EMP
       WHERE WORKDEPT = X.WORKDEPT);
```

The following example demonstrates the use of a correlated reference in the select list of a subquery:

```
UPDATE BP1TBL T1
  SET (KEY1, CHAR1, VCHAR1) =
    (SELECT VALUE(T2.KEY1,T1.KEY1), VALUE(T2.CHAR1,T1.CHAR1),
            VALUE(T2.VCHAR1,T1.VCHAR1)
       FROM BP2TBL T2
       WHERE (T2.KEY1 = T1.KEY1))
  WHERE KEY1 IN
    (SELECT KEY1
       FROM BP2TBL T3
       WHERE KEY2 > 0);
```

Using correlated subqueries in an UPDATE statement:

Use correlation names in an UPDATE statement to refer to the rows that you are updating. The subquery for which you specified a correlation name is called a *correlated subquery*.

For example, when all activities of a project must complete before September 2006, your department considers that project to be a priority project. Assume that you have added the PRIORITY column to DSN8B10.PROJ. You can use the following SQL statement to evaluate the projects in the DSN8B10.PROJ table, and write a 1 (a flag to indicate PRIORITY) in the PRIORITY column for each priority project:

```
UPDATE DSN8B10.PROJ X
SET PRIORITY = 1
WHERE DATE('2006-09-01') >
  (SELECT MAX(ACENDATE)
     FROM DSN8B10.PROJACT
     WHERE PROJNO = X.PROJNO);
```

As Db2 examines each row in the DSN8B10.PROJ table, it determines the maximum activity end date (the ACENDATE column) for all activities of the project (from the DSN8B10.PROJACT table). If the end date of each activity that is associated with the project is before September 2006, the current row in the DSN8B10.PROJ table qualifies, and Db2 updates it.

Using correlated subqueries in a DELETE statement:

Use correlation names in a DELETE statement to refer to the rows that you are deleting. The subquery for which you specified a correlation name is called a *correlated subquery*. Db2 evaluates the correlated subquery once for each row in the table that is named in the DELETE statement to decide whether to delete the row.

Using tables with no referential constraints:

Suppose that a department considers a project to be complete when the combined amount of time currently spent on it is less than or equal to half of a person's time. The department then deletes the rows for that project from the DSN8B10.PROJ table. In the examples in this topic, PROJ and PROJACT are independent tables; that is, they are separate tables with no referential constraints defined on them.

```
DELETE FROM DSN8B10.PROJ X
  WHERE .5 >
    (SELECT SUM(ACSTAFF)
       FROM DSN8B10.PROJACT
       WHERE PROJNO = X.PROJNO);
```

To process this statement, Db2 determines for each project (represented by a row in the DSN8B10.PROJ table) whether the combined staffing for that project is less than 0.5. If it is, Db2 deletes that row from the DSN8B10.PROJ table.

To continue this example, suppose that Db2 deletes a row in the DSN8B10.PROJ table. You must also delete rows that are related to the deleted project in the DSN8B10.PROJACT table. To do this, use a statement similar to this statement:

```
DELETE FROM DSN8B10.PROJACT X
  WHERE NOT EXISTS
    (SELECT *
       FROM DSN8B10.PROJ
       WHERE PROJNO = X.PROJNO);
```

Db2 determines, for each row in the DSN8B10.PROJACT table, whether a row with the same project number exists in the DSN8B10.PROJ table. If not, Db2 deletes the row from DSN8B10.PROJACT.

Using a single table:

A subquery of a searched DELETE statement (a DELETE statement that does not use a cursor) can reference the same table from which rows are deleted. In the following statement, which deletes the employee with the highest salary from each department, the employee table appears in the outer DELETE and in the subselect:

```
DELETE FROM YEMP X
  WHERE SALARY = (SELECT MAX(SALARY) FROM YEMP Y
                  WHERE X.WORKDEPT =Y.WORKDEPT);
```

This example uses a copy of the employee table for the subquery.

The following statement, without a correlated subquery, yields equivalent results:

```
DELETE FROM YEMP
  WHERE (SALARY, WORKDEPT) IN (SELECT MAX(SALARY), WORKDEPT
                               FROM YEMP
                               GROUP BY WORKDEPT);
```

Using tables with referential constraints:

Db2 restricts delete operations for dependent tables that are involved in referential constraints. If a DELETE statement has a subquery that references a table that is involved in the deletion, make the last delete rule in the path to that table RESTRICT or NO ACTION. This action ensures that the result of the subquery is not materialized before the deletion occurs. However, if the result of the subquery is materialized before the deletion, the delete rule can also be CASCADE or SET NULL.

Example: Without referential constraints, the following statement deletes departments from the department table whose managers are not listed correctly in the employee table:

```
DELETE FROM DSN8B10.DEPT THIS
  WHERE NOT DEPTNO =
    (SELECT WORKDEPT
       FROM DSN8B10.EMP
       WHERE EMPNO = THIS.MGRNO);
```

With the referential constraints that are defined for the sample tables, this statement causes an error because the result table for the subquery is not materialized before the deletion occurs. Because DSN8B10.EMP is a dependent table of DSN8B10.DEPT, the deletion involves the table that is referred to in the subquery, and the last delete rule in the path to EMP is SET NULL, not RESTRICT or NO ACTION. If the statement could execute, its results would depend on the order in which Db2 accesses the rows. Therefore, Db2 prohibits the deletion.

Restrictions when using distinct types with UNION, EXCEPT, and INTERSECT

Db2 enforces strong typing of distinct types with UNION, EXCEPT, and INTERSECT. When you use these keywords to combine column values from several tables, the combined columns must be of the same types. If a column is a distinct type, the corresponding column must be the same distinct type.

Example: Suppose that you create a view that combines the values of the US_SALES, EUROPEAN_SALES, and JAPAN_SALES tables. The TOTAL columns in the three tables are of different distinct types. Before you combine the table values, you must convert the types of two of the TOTAL columns to the type of the third TOTAL column. Assume that the US_DOLLAR type has been chosen as the common distinct type. Because Db2 does not generate cast functions to convert from one distinct type to another, two user-defined functions must exist:

- A function called EURO_TO_US that converts values of type EURO to type US_DOLLAR
- A function called YEN_TO_US that converts values of type JAPANESE_YEN to type US_DOLLAR

Then you can execute a query like this to display a table of combined sales:

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
FROM US_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, EURO_TO_US(TOTAL)
FROM EUROPEAN_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, YEN_TO_US(TOTAL)
FROM JAPAN_SALES;
```

Because the result type of both the YEN_TO_US function and the EURO_TO_US function is US_DOLLAR, you have satisfied the requirement that the distinct types of the combined columns are the same.

Comparison of distinct types

You can compare an object with a distinct type only to an object with exactly the same distinct type. You cannot compare data of a distinct type directly to data of its source type. However, you can compare a distinct type to its source type by using a cast function.

The basic rule for comparisons is that the data types of the operands must be compatible. The compatibility rule defines, for example, that all numeric types (SMALLINT, INTEGER, FLOAT, and DECIMAL) are compatible. That is, you can compare an INTEGER value with a value of type FLOAT. However, you cannot compare an object of a distinct type to an object of a different type. You can compare an object with a distinct type only to an object with exactly the same distinct type.

For example, suppose you want to know which products sold more than $100 000.00 in the US in the month of July in 2003 (7/03). Because you cannot compare data of type US_DOLLAR with instances of data of the source type of US_DOLLAR (DECIMAL) directly, you must use a cast function to cast data from DECIMAL to US_DOLLAR or from US_DOLLAR to DECIMAL. Whenever you create a distinct type, Db2 creates two cast functions, one to cast from the source type to the distinct type and the other to cast from the distinct type to the source type. For distinct type US_DOLLAR, Db2 creates a cast function called DECIMAL and a cast function called US_DOLLAR. When you compare an object of type US_DOLLAR to an object of type DECIMAL, you can use one of those cast functions to make the data types identical for the comparison. Suppose table US_SALES is defined like this:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1990),
   TOTAL         US_DOLLAR);
```

Then you can cast DECIMAL data to US_DOLLAR like this:

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR(100000.00)
  AND    MONTH = 7
  AND    YEAR  = 2003;
```

The casting satisfies the requirement that the compared data types are identical.

You cannot use host variables in statements that you prepare for dynamic execution. As explained in "Dynamically executing an SQL statement by using PREPARE and EXECUTE" on page 520, you can substitute parameter markers for host variables when you prepare a statement, and then use host variables when you execute the statement.

If you use a parameter marker in a predicate of a query, and the column to which you compare the value represented by the parameter marker is of a distinct type, you must cast the parameter marker to the distinct type, or cast the column to its source type.

For example, suppose that distinct type CNUM is defined like this:

```
CREATE DISTINCT TYPE CNUM AS INTEGER;
```

Table CUSTOMER is defined like this:

```
CREATE TABLE CUSTOMER
  (CUST_NUM     CNUM NOT NULL,
   FIRST_NAME   CHAR(30) NOT NULL,
   LAST_NAME    CHAR(30) NOT NULL,
   PHONE_NUM    CHAR(20) WITH DEFAULT,
   PRIMARY KEY  (CUST_NUM));
```

In an application program, you prepare a SELECT statement that compares the CUST_NUM column to a parameter marker. Because CUST_NUM is of a distinct type, you must cast the distinct type to its source type:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
  WHERE CAST(CUST_NUM AS INTEGER) = ?
```

Alternatively, you can cast the parameter marker to the distinct type:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
  WHERE CUST_NUM = CAST (? AS CNUM)
```

Nested SQL statements

An SQL statement can explicitly invoke user-defined functions or stored procedures or can implicitly activate triggers that invoke user-defined functions or stored procedures. This situation is known as nesting of SQL statements. Db2 supports as many as to 64 levels of nesting.

The following example shows SQL statement nesting.

```
Trigger TR1 is defined on table T3:
 CREATE TRIGGER TR1
  AFTER UPDATE ON T3
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
   CALL SP3(PARM1);
  END
Program P1 (nesting level 1) contains:
 SELECT UDF1(C1) FROM T1;
UDF1 (nesting level 2) contains:
 CALL SP2(C2);
SP2 (nesting level 3) contains:
 UPDATE T3 SET C3=1;
SP3 (nesting level 4) contains:
 SELECT UDF4(C4) FROM T4;
⋮
SP16 (nesting level 16) cannot invoke stored procedures
 or user-defined functions
```

Be aware of the following Db2 restrictions on nested SQL statements:

- Restrictions for SELECT statements:

  When you execute a SELECT statement on a table, you cannot execute INSERT, UPDATE, MERGE, or DELETE statements on the same table at a lower level of nesting.

  For example, suppose that you execute this SQL statement at level 1 of nesting:

  ```
  SELECT UDF1(C1) FROM T1;
  ```

  You cannot execute this SQL statement at a lower level of nesting:

  ```
  INSERT INTO T1 VALUES(…);
  ```

- Restrictions for SELECT FROM FINAL TABLE statements that specify INSERT, UPDATE, or DELETE statements to change data:

  When you execute this type of statement, an error occurs if both of the following conditions exist:

  – The SELECT statement that modifies data (by specifying INSERT, UPDATE, or DELETE) activates an AFTER TRIGGER.
  – The AFTER TRIGGER results in additional nested SQL operations that modify the table that is the target of the original SELECT statement that modifies data.

- Restrictions for INSERT, UPDATE, MERGE, and DELETE statements:

  When you execute an INSERT, UPDATE, MERGE, or DELETE statement on a table, you cannot access that table from a user-defined function or stored procedure that is at a lower level of nesting.

  For example, suppose that you execute this SQL statement at level 1 of nesting:

  ```
  DELETE FROM T1 WHERE UDF3(T1.C1) = 3;
  ```

You cannot execute this SELECT statement at a lower level of nesting:

```
SELECT * FROM T1;
```

If the AFTER trigger is not activated by an INSERT, UPDATE, or DELETE data change statement that is specified in a data-change-table-reference SELECT FROM FINAL TABLE, the preceding list of restrictions do not apply to SQL statements that are executed at a lower level of nesting as a result of an after trigger. For example, suppose an UPDATE statement at nesting level 1 activates an after update trigger, which calls a stored procedure. The stored procedure executes two SQL statements that reference the triggering table: one SELECT statement and one INSERT statement. In this situation, both the SELECT and the INSERT statements can be executed even though they are at nesting level 3.

Although trigger activations count in the levels of SQL statement nesting, the previous restrictions on SQL statements do not apply to SQL statements that are executed in the trigger body.

Example: Suppose that trigger TR1 is defined on table T1:

```
CREATE TRIGGER TR1
 AFTER INSERT ON T1
 FOR EACH STATEMENT MODE DB2SQL
 BEGIN ATOMIC
  UPDATE T1 SET C1=1;
 END
```

Now suppose that you execute this SQL statement at level 1 of nesting:

```
INSERT INTO T1 VALUES(…);
```

Although the UPDATE statement in the trigger body is at level 2 of nesting and modifies the same table that the triggering statement updates, Db2 can execute the INSERT statement successfully.

Retrieving a set of rows by using a cursor

In an application program, you can retrieve a set of rows from a table or a result table that is returned by a stored procedure. You can retrieve one or more rows at a time.

About this task

Use either of the following types of cursors to retrieve rows from a result table:

- A row-positioned cursor retrieves at most a single row at a time from the result table into host variables. At any point in time, the cursor is positioned on at most a single row. For information about how to use a row-positioned cursor, see "Accessing data by using a row-positioned cursor" on page 405.
- A rowset-positioned cursor retrieves zero, one, or more rows at a time, as a rowset, from the result table into host-variable arrays. At any point in time, the cursor can be positioned on a rowset. You can reference all of the rows in the rowset, or only one row in the rowset, when you use a positioned DELETE or positioned UPDATE statement. For information about how to use a rowset-positioned cursor, see "Accessing data by using a rowset-positioned cursor" on page 409.

Cursors

A *cursor* is a mechanism that points to one or more rows in a set of rows. The rows are retrieved from a table or in a result set that is returned by a stored procedure. Your application program can use a cursor to retrieve rows from a table.

About this task

Cursors bound with cursor stability that are used in block fetch operations are particularly vulnerable to reading data that has already changed. In a block fetch, database access prefetches rows ahead of the row retrieval controlled by the application. During that time the cursor might close, and the locks might be released, before the application receives the data. Thus, it is possible for the application to fetch a row of values that no longer exists, or to miss a recently inserted row. In many cases, that is acceptable; a case for which it is **not** acceptable is said to require *data currency*.

If your application requires data currency for a cursor, you need to prevent block fetching for the data to which it points. To prevent block fetching for a distributed cursor, declare the cursor with the FOR UPDATE clause.

Types of cursors

You can declare row-positioned or rowset-positioned cursors in a number of ways. These cursors can be scrollable or not scrollable, held or not held, or returnable or not returnable.

In addition, you can declare a returnable cursor in a stored procedure by including the WITH RETURN clause; the cursor can return result sets to a caller of the stored procedure.

Scrollable and non-scrollable cursors:

When you declare a cursor, you tell Db2 whether you want the cursor to be scrollable or non-scrollable by including or omitting the SCROLL clause. This clause determines whether the cursor moves sequentially forward through the result table or can move randomly through the result table.

Using a non-scrollable cursor:

The simplest type of cursor is a non-scrollable cursor. A non-scrollable cursor can be either row-positioned or rowset-positioned. A row-positioned non-scrollable cursor moves forward through its result table one row at a time. Similarly, a rowset-positioned non-scrollable cursor moves forward through its result table one rowset at a time.

A non-scrollable cursor always moves sequentially forward in the result table. When the application opens the cursor, the cursor is positioned before the first row (or first rowset) in the result table. When the application executes the first FETCH, the cursor is positioned on the first row (or first rowset). When the application executes subsequent FETCH statements, the cursor moves one row ahead (or one rowset ahead) for each FETCH. After each FETCH statement, the cursor is positioned on the row (or rowset) that was fetched.

After the application executes a positioned UPDATE or positioned DELETE statement, the cursor stays at the current row (or rowset) of the result table. You cannot retrieve rows (or rowsets) backward or move to a specific position in a result table with a non-scrollable cursor.

Using a scrollable cursor:

To make a cursor scrollable, you declare it as scrollable. A scrollable cursor can be either row-positioned or rowset-positioned. To use a scrollable cursor, you execute FETCH statements that indicate where you want to position the cursor.

If you want to order the rows of the cursor's result set, and you also want the cursor to be updatable, you need to declare the cursor as scrollable, even if you use it only to retrieve rows (or rowsets) sequentially. You can use the ORDER BY clause in the declaration of an updatable cursor only if you declare the cursor as scrollable.

Declaring a scrollable cursor:

To indicate that a cursor is scrollable, you declare it with the SCROLL keyword. The following examples show a characteristic of scrollable cursors: the *sensitivity*.

The following figure shows a declaration for an insensitive scrollable cursor.

```
EXEC SQL DECLARE C1 INSENSITIVE SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8B10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declaring a scrollable cursor with the INSENSITIVE keyword has the following effects:

- The size, the order of the rows, and the values for each row of the result table do not change after the application opens the cursor.

  Rows that are inserted into the underlying table are not added to the result table.
- The result table is read-only. Therefore, you cannot declare the cursor with the FOR UPDATE clause, and you cannot use the cursor for positioned update or delete operations.

The following figure shows a declaration for a sensitive static scrollable cursor.

```
EXEC SQL DECLARE C2 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8B10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declaring a cursor as SENSITIVE STATIC has the following effects:

- The size of the result table does not grow after the application opens the cursor.

  Rows that are inserted into the underlying table are not added to the result table.
- The order of the rows does not change after the application opens the cursor.

  If the cursor declaration contains an ORDER BY clause, and the columns that are in the ORDER BY clause are updated after the cursor is opened, the order of the rows in the result table does not change.
- When the application executes positioned UPDATE and DELETE statements with the cursor, those changes are visible in the result table.
- When the current value of a row no longer satisfies the SELECT statement that was used in the cursor declaration, that row is no longer visible in the result table.
- When a row of the result table is deleted from the underlying table, that row is no longer visible in the result table.
- Changes that are made to the underlying table by other cursors or other application processes can be visible in the result table, depending on whether the FETCH statements that you use with the cursor are FETCH INSENSITIVE or FETCH SENSITIVE statements.

The following figure shows a declaration for a sensitive dynamic scrollable cursor.

```
EXEC SQL DECLARE C2 SENSITIVE DYNAMIC SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8B10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declaring a cursor as SENSITIVE DYNAMIC has the following effects:

- The size and contents of the result table can change with every fetch.

  The base table can change while the cursor is scrolling on it. If another application process changes the data, the cursor sees the newly changed data when it is committed. If the application process of the cursor changes the data, the cursor sees the newly changed data immediately.
- The order of the rows can change after the application opens the cursor.

  If the SELECT statement of the cursor declaration contains an ORDER BY clause, and columns that are in the ORDER BY clause are updated after the cursor is opened, the order of the rows in the result table changes.
- When the application executes positioned UPDATE and DELETE statements with the cursor, those changes are visible. In addition, when the application executes insert, update, or delete operations (within the application but outside the cursor), those changes are visible.
- All committed inserts, updates, and deletes by other application processes are visible.

• Because the FETCH statement executes against the base table, the cursor needs no temporary result table. When you define a cursor as SENSITIVE DYNAMIC, you cannot specify the INSENSITIVE keyword in a FETCH statement for that cursor.

Visibility of changes to a result table:

Whether a cursor can view its own changes or the changes that are made to the data by other processes or cursors depends on how the cursor is declared, and the updatability of the cursor. Visibility also depends on the type of fetch operation that is executed with the cursor. The following table summarizes the visibility of changes to a result table for each type of cursor.

Declared cursor type	Cursor is updatable or read-only?	Changes by the cursor are visible in the result table?"3" on page 403	Changes by other cursors or processes are visible to the result table?
NO SCROLL (result table is materialized)	Read-only"1" on page 403	Not applicable	No
NO SCROLL (result table is not materialized)	Updatable"2" on page 403	Yes	Yes
INSENSITIVE SCROLL	Read-only"4" on page 403	Not applicable	No
SENSITIVE STATIC SCROLL	Updatable"2" on page 403, "6" on page 403	Yes	Depends on the explicitly or implicitly specified sensitivity in the FETCH clause"5" on page 403
SENSITIVE DYNAMIC SCROLL	Updatable"2" on page 403	Yes	Yes"7" on page 403

Notes:

1. The content of the SELECT statement of the cursor makes the cursor implicitly read-only.
2. The cursor is updatable only if FOR READ ONLY or FOR FETCH ONLY is not specified as part of the SELECT statement of the cursor, and there is nothing in the content of the SELECT statement makes the cursor implicitly read-only.
3. If INSENSITIVE is specified on FETCH, only changes made by the same cursor are visible, assuming that the rows being fetched have not already been read by a SENSITIVE FETCH on the same cursor.
4. An INSENSITIVE cursor is read-only if an updatability clause is not specified.
5. The sensitivity clause in a FETCH statement affects the visibility of others' changes as follows:
   - For FETCH INSENSITIVE: Only positioned updates and deletes that are made by the same cursor are visible.
   - For FETCH SENSITIVE: All updates and deletes are visible.
6. Positioned updates and deletes are disallowed if the values of the selected columns do not match the current values of the columns in the base table, even if the row satisfies the predicate of the SELECT statement of the cursor.
7. All updates and deletes that are made by this cursor, and committed changes that are made by other processes are visible on subsequent FETCH statements. Inserts that are made by this process are also be visible as the result table is scrolled. Inserts by other processes into the base tables underlying the result table are visible after they are committed.

Related concepts

FETCH statement interaction between row and rowset positioning

When you declare a cursor with the WITH ROWSET POSITIONING clause, you can intermix row-positioned FETCH statements with rowset-positioned FETCH statements.

Comparison of scrollable cursors
Whether a scrollable cursor can view the changes that are made to the data by other processes or cursors depends on how the cursor is declared. It also depends on the type of fetch operation that is executed.

Held and non-held cursors

A held cursor does not close after a commit operation. A cursor that is not held closes after a commit operation. You specify whether you want a cursor to be held or not held by including or omitting the WITH HOLD clause when you declare the cursor.

After a commit operation, the position of a held cursor depends on its type:

- A non-scrollable cursor that is held is positioned after the last retrieved row and before the next logical row. The next row can be returned from the result table with a FETCH NEXT statement.
- A static scrollable cursor that is held is positioned on the last retrieved row. The last retrieved row can be returned from the result table with a FETCH CURRENT statement.
- A dynamic scrollable cursor that is held is positioned after the last retrieved row and before the next logical row. Use a FETCH statement to reposition the cursor to retrieve the desired row or rowset. Db2 returns SQLCODE +231 for a FETCH statement that specifies the CURRENT keyword for a single-row fetch.

A held cursor can close when:

- You issue a CLOSE cursor, ROLLBACK, or CONNECT statement
- You issue a CAF CLOSE function call or an RRSAF TERMINATE THREAD function call
- The application program terminates.

If the program abnormally terminates, the cursor position is lost. To prepare for restart, your program must reposition the cursor.

The following restrictions apply to cursors that are declared WITH HOLD:

- Do not use DECLARE CURSOR WITH HOLD with the new user signon from a Db2 attachment facility, because all open cursors are closed.
- Do not declare a WITH HOLD cursor in a thread that might become inactive. If you do, its locks are held indefinitely.

IMS

You **cannot** use DECLARE CURSOR...WITH HOLD in message processing programs (MPP) and message-driven batch message processing (BMP). Each message is a new user for Db2; whether or not you declare them using WITH HOLD, no cursors continue for new users. You can use WITH HOLD in non-message-driven BMP and DL/I batch programs.

CICS

In CICS applications, you can use DECLARE CURSOR...WITH HOLD to indicate that a cursor should not close at a commit or sync point. However, SYNCPOINT ROLLBACK closes all cursors, and end-of-task (EOT) closes all cursors before Db2 reuses or terminates the thread. Because pseudo-conversational transactions usually have multiple EXEC CICS RETURN statements and thus span multiple EOTs, the scope of a held cursor is limited. Across EOTs, you must reopen and reposition a cursor declared WITH HOLD, as if you had not specified WITH HOLD.

You should always close cursors that you no longer need. If you let Db2 close a CICS attachment cursor, the cursor might not close until the CICS attachment facility reuses or terminates the thread.

If the CICS application is using a protected entry thread, this thread will continue to hold resources, even when the task that has used these resources ends. These resources will not be released until the protected thread terminates.

The following cursor declaration causes the cursor to maintain its position in the DSN8B10.EMP table after a commit point:

```
EXEC SQL
  DECLARE EMPLUPDT CURSOR WITH HOLD FOR
    SELECT EMPNO, LASTNAME, PHONENO, JOB, SALARY, WORKDEPT
      FROM DSN8B10.EMP
      WHERE WORKDEPT < 'D11'
      ORDER BY EMPNO
END-EXEC.
```

Accessing data by using a row-positioned cursor

A row-positioned cursor is a cursor that points to a single row and retrieves at most a single row at a time from the result table. You can specify a fetch request to specify which rows to retrieve, relative to the current cursor position.

Procedure

To access data by using a row-positioned cursor:

1. Execute a DECLARE CURSOR statement to define the result table on which the cursor operates. See "Declaring a row cursor" on page 405.
2. Execute an OPEN CURSOR to make the cursor available to the application. See "Opening a row cursor" on page 407.
3. Specify what the program is to do when all rows have been retrieved. See "Specifying the action that the row cursor is to take when it reaches the end of the data" on page 407.
4. Execute multiple SQL statements to retrieve data from the table or modify selected rows of the table. See "Executing SQL statements by using a row cursor" on page 408.
5. Execute a CLOSE CURSOR statement to make the cursor unavailable to the application. See "Closing a row cursor" on page 409.

Results

Your program can have several cursors, each of which performs the previous steps.

Declaring a row cursor

Before you can use a row-positioned cursor to retrieve rows, you must declare the cursor. When you declare a cursor, you identify a set of rows that are to be accessed with the cursor.

Procedure

To declare a row cursor, issue a DECLARE CURSOR statement.

The DECLARE CURSOR statement names a cursor and specifies a SELECT statement. The SELECT statement defines the criteria for the rows that are to make up the result table.

The following example shows a simple form of the DECLARE CURSOR statement:

```
EXEC SQL
  DECLARE C1 CURSOR FOR
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
      FROM DSN8B10.EMP
  END-EXEC.
```

You can use this cursor to list select information about employees.

More complicated cursors might include WHERE clauses or joins of several tables. For example, suppose that you want to use a cursor to list employees who work on a certain project. Declare a cursor like this to identify those employees:

```
EXEC SQL
  DECLARE C2 CURSOR FOR
```

```
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
      FROM DSN8B10.EMP X
      WHERE EXISTS
        (SELECT *
           FROM DSN8B10.PROJ Y
           WHERE X.EMPNO=Y.RESPEMP
           AND Y.PROJNO=:GOODPROJ);
```

Declaring cursors for tables that use multilevel security

You can declare a cursor that retrieves rows from a table that uses multilevel security with row-level granularity. However, the result table for the cursor contains only those rows that have a security label value that is equivalent to or dominated by the security label value of your ID.

Updating a column

You can update columns in the rows that you retrieve. Updating a row after you use a cursor to retrieve it is called a *positioned* update. If you intend to perform any positioned updates on the identified table, include the FOR UPDATE clause. The FOR UPDATE clause has two forms:

- The first form is FOR UPDATE OF *column-list*. Use this form when you know in advance which columns you need to update.
- The second form is FOR UPDATE, with no column list. Use this form when you might use the cursor to update any of the columns of the table.

For example, you can use this cursor to update only the SALARY column of the employee table:

```
EXEC SQL
  DECLARE C1 CURSOR FOR
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
      FROM DSN8B10.EMP X
      WHERE EXISTS
        (SELECT *
           FROM DSN8B10.PROJ Y
           WHERE X.EMPNO=Y.RESPEMP
           AND Y.PROJNO=:GOODPROJ)
    FOR UPDATE OF SALARY;
```

If you might use the cursor to update any column of the employee table, define the cursor like this:

```
EXEC SQL
  DECLARE C1 CURSOR FOR
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
      FROM DSN8B10.EMP X
      WHERE EXISTS
        (SELECT *
           FROM DSN8B10.PROJ Y
           WHERE X.EMPNO=Y.RESPEMP
           AND Y.PROJNO=:GOODPROJ)
    FOR UPDATE;
```

Db2 must do more processing when you use the FOR UPDATE clause without a column list than when you use the FOR UPDATE clause with a column list. Therefore, if you intend to update only a few columns of a table, your program can run more efficiently if you include a column list.

The precompiler options NOFOR and STDSQL affect the use of the FOR UPDATE clause in static SQL statements. If you do not specify the FOR UPDATE clause in a DECLARE CURSOR statement, and you do not specify the STDSQL(YES) option or the NOFOR precompiler options, you receive an error if you execute a positioned UPDATE statement.

You can update a column of the identified table even though it is not part of the result table. In this case, you do not need to name the column in the SELECT statement. When the cursor retrieves a row (using FETCH) that contains a column value you want to update, you can use UPDATE ... WHERE CURRENT OF to identify the row that is to be updated.

Read-only result table

Some result tables cannot be updated—for example, the result of joining two or more tables.

Related concepts

Multilevel security (Managing Security)

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

DECLARE CURSOR (Db2 SQL)

select-statement (Db2 SQL)

Opening a row cursor

After you declare a row cursor, you need to tell Db2 that you are ready to process the first row of the result table. This action is called opening the cursor.

About this task

To open a row cursor, execute the OPEN statement in your program. Db2 then uses the SELECT statement within DECLARE CURSOR to identify a set of rows. If you use host variables in the search condition of that SELECT statement, Db2 uses the **current value** of the variables to select the rows. The result table that satisfies the search condition might contain zero, one, or many rows. An example of an OPEN statement is:

```
EXEC SQL
  OPEN C1
END-EXEC.
```

If you use the CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP special registers in a cursor, Db2 determines the values in those special registers only when it opens the cursor. Db2 uses the values that it obtained at OPEN time for all subsequent FETCH statements.

Two factors that influence the amount of time that Db2 requires to process the OPEN statement are:

- Whether Db2 must perform any sorts before it can retrieve rows
- Whether Db2 uses parallelism to process the SELECT statement of the cursor

Specifying the action that the row cursor is to take when it reaches the end of the data

Your program must be coded to recognize and handle an end-of-data condition whenever you use a row cursor to fetch a row.

About this task

To determine whether the program has retrieved the last row of data, test the SQLCODE field for a value of 100 or the SQLSTATE field for a value of '02000'. These codes occur when a FETCH statement has retrieved the last row in the result table and your program issues a subsequent FETCH. For example:

```
IF SQLCODE = 100 GO TO DATA-NOT-FOUND.
```

An alternative to this technique is to code the WHENEVER NOT FOUND statement. The WHENEVER NOT FOUND statement causes your program to branch to another part that then issues a CLOSE statement. For example, to branch to label DATA-NOT-FOUND when the FETCH statement does not return a row, use this statement:

```
EXEC SQL
  WHENEVER NOT FOUND GO TO DATA-NOT-FOUND
END-EXEC.
```

For more information about the WHENEVER NOT FOUND statement, see “Checking the execution of SQL statements” on page 527.

Executing SQL statements by using a row cursor

You can use row cursors to execute FETCH statements, positioned UPDATE statements, and positioned DELETE statements.

About this task

Execute a FETCH statement for one of the following purposes:

- To copy data from a row of the result table into one or more host variables
- To position the cursor before you perform a positioned update or positioned delete operation

The following example shows a FETCH statement that retrieves selected columns from the employee table:

```
EXEC SQL
  FETCH C1 INTO
    :HV-EMPNO, :HV-FIRSTNME, :HV-MIDINIT, :HV-LASTNAME, :HV-SALARY :IND-SALARY
END-EXEC.
```

The SELECT statement within DECLARE CURSOR statement identifies the result table from which you fetch rows, but Db2 does not retrieve any data until your application program executes a FETCH statement.

When your program executes the FETCH statement, Db2 positions the cursor on a row in the result table. That row is called the *current row*. Db2 then copies the current row contents into the program host variables that you specify on the INTO clause of FETCH. This sequence repeats each time you issue FETCH, until you process all rows in the result table.

The row that Db2 points to when you execute a FETCH statement depends on whether the cursor is declared as a scrollable or non-scrollable.

When you query a remote subsystem with FETCH, consider using block fetch for better performance. Block fetch processes rows ahead of the current row. You cannot use a block fetch when you perform a positioned update or delete operation.

After your program has executed a FETCH statement to retrieve the current row, you can use a positioned UPDATE statement to modify the data in that row. An example of a positioned UPDATE statement is:

```
EXEC SQL
  UPDATE DSN8B10.EMP
    SET SALARY = 50000
    WHERE CURRENT OF C1
END-EXEC.
```

A positioned UPDATE statement updates the row on which the cursor is positioned.

A positioned UPDATE statement is subject to these restrictions:

- You cannot update a row if your update violates any unique, check, or referential constraints.
- You cannot use an UPDATE statement to modify the rows of a created temporary table. However, you can use an UPDATE statement to modify the rows of a declared temporary table.
- If the right side of the SET clause in the UPDATE statement contains a fullselect, that fullselect cannot include a correlated name for a table that is being updated.
- You cannot use an SQL data change statement in the FROM clause of a SELECT statement that defines a cursor that is used in a positioned UPDATE statement.
- A positioned UPDATE statement will fail if the value of the security label column of the row where the cursor is positioned is not equivalent to the security label value of your user id. If your user id has write down privilege, a positioned UPDATE statement will fail if the value of the security label column of the row where the cursor is positioned does not dominate the security label value of your user id.

After your program has executed a FETCH statement to retrieve the current row, you can use a positioned DELETE statement to delete that row. A example of a positioned DELETE statement looks like this:

```
EXEC SQL
  DELETE FROM DSN8B10.EMP
    WHERE CURRENT OF C1
END-EXEC.
```

A positioned DELETE statement deletes the row on which the cursor is positioned.

A positioned DELETE statement is subject to these restrictions:

- You cannot use a DELETE statement with a cursor to delete rows from a created temporary table. However, you can use a DELETE statement with a cursor to delete rows from a declared temporary table.
- After you have deleted a row, you cannot update or delete another row using that cursor until you execute a FETCH statement to position the cursor on another row.
- You cannot delete a row if doing so violates any referential constraints.
- You cannot use an SQL data change statement in the FROM clause of a SELECT statement that defines a cursor that is used in a positioned DELETE statement.
- A positioned DELETE statement will fail if the value of the security label column of the row where the cursor is positioned is not equivalent to the security label value of your user id. If your user id has write down privilege, a positioned DELETE statement will fail if the value of the security label column of the row where the cursor is positioned does not dominate the security label value of your user id.

Closing a row cursor

Close a row cursor when it finishes processing rows if you want to free the resources or if you want to use the cursor again. Otherwise, you can let Db2 automatically close the cursor when the current transaction terminates or when your program terminates.

About this task

To free the resources that are held by the cursor, close the cursor explicitly by issuing the CLOSE statement.

If you want to use the rowset cursor again, reopen it.

Procedure

Issue a CLOSE statement.
An example of a CLOSE statement looks like this:

```
EXEC SQL
  CLOSE C1
END-EXEC.
```

Accessing data by using a rowset-positioned cursor

A rowset-positioned cursor is a cursor that can return one or more rows for a single fetch operation. The cursor is positioned on the set of rows that are to be fetched.

Procedure

To access data by using a rowset-positioned cursor:

1. Execute a DECLARE CURSOR statement to define the result table on which the cursor operates. See "Declaring a rowset cursor" on page 410.
2. Execute an OPEN CURSOR to make the cursor available to the application. See "Opening a rowset cursor" on page 410.
3. Specify what the program is to do when all rows have been retrieved. See "Specifying the action that the rowset cursor is to take when it reaches the end of the data" on page 410.

4. Execute multiple SQL statements to retrieve data from the table or modify selected rows of the table. See "Executing SQL statements by using a rowset cursor" on page 411.
5. Execute a CLOSE CURSOR statement to make the cursor unavailable to the application. See "Closing a rowset cursor" on page 414.

Results

Your program can have several cursors, each of which performs the previous steps.

Declaring a rowset cursor

Before you can use a rowset-positioned cursor to retrieve rows, you must declare a cursor that is enabled to fetch rowsets. When you declare a cursor, you identify a set of rows that are to be accessed with the cursor.

About this task

For restrictions that apply to rowset-positioned cursors and row-positioned cursors, see "Declaring a row cursor" on page 405.

Procedure

Use the WITH ROWSET POSITIONING clause in the DECLARE CURSOR statement.
The following example shows how to declare a rowset cursor:

```
EXEC SQL
  DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR
    SELECT EMPNO, LASTNAME, SALARY
      FROM DSN8B10.EMP
END-EXEC.
```

Opening a rowset cursor

After you declare a rowset cursor, you need to tell Db2 that you are ready to process the first rowset of the result table. This action is called opening the cursor.

About this task

To open a rowset cursor, execute the OPEN statement in your program. Db2 then uses the SELECT statement within DECLARE CURSOR to identify the rows in the result table. For more information about the OPEN CURSOR process, see "Opening a row cursor" on page 407.

Specifying the action that the rowset cursor is to take when it reaches the end of the data

Your program must be coded to recognize and handle an end-of-data condition whenever you use a rowset cursor to fetch rows.

About this task

To determine whether the program has retrieved the last row of data in the result table, test the SQLCODE field for a value of +100 or the SQLSTATE field for a value of '02000'. With a rowset cursor, these codes occur when a FETCH statement retrieves the last row in the result table. However, when the last row has been retrieved, the program must still process the rows in the last rowset through that last row. For an example of end-of-data processing for a rowset cursor, see "Examples of fetching rows by using cursors" on page 427.

To determine the number of retrieved rows, use either of the following values:

- The contents of the SQLERRD(3) field in the SQLCA

- The contents of the ROW_COUNT item of GET DIAGNOSTICS

For information about GET DIAGNOSTICS, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

If you declare the cursor as dynamic scrollable, and SQLCODE has the value +100, you can continue with a FETCH statement until no more rows are retrieved. Additional fetches might retrieve more rows because a dynamic scrollable cursor is sensitive to updates by other application processes. For information about dynamic cursors, see "Types of cursors" on page 401.

Executing SQL statements by using a rowset cursor

You can use rowset cursors to execute multiple-row FETCH statements, positioned UPDATE statements, and positioned DELETE statements.

About this task

You can execute these static SQL statements when you use a rowset cursor:

- A multiple-row FETCH statement that copies a rowset of column values into either of the following data areas:
  - Host-variable arrays that are declared in your program
  - Dynamically-allocated arrays whose storage addresses are put into an SQL descriptor area (SQLDA), along with the attributes of the columns that are to be retrieved
- After either form of the multiple-row FETCH statement, you can issue:
  - A positioned UPDATE statement on the current rowset
  - A positioned DELETE statement on the current rowset

You must use the WITH ROWSET POSITIONING clause of the DECLARE CURSOR statement if you plan to use a rowset-positioned FETCH statement.

The following example shows a FETCH statement that retrieves 20 rows into host-variable arrays that are declared in your program:

```
EXEC SQL
  FETCH NEXT ROWSET FROM C1
  FOR 20 ROWS
  INTO :HVA-EMPNO, :HVA-LASTNAME, :HVA-SALARY :INDA-SALARY
END-EXEC.
```

When your program executes a FETCH statement with the ROWSET keyword, the cursor is positioned on a rowset in the result table. That rowset is called the *current rowset*. The dimension of each of the host-variable arrays must be greater than or equal to the number of rows to be retrieved.

Suppose that you want to dynamically allocate the storage needed for the arrays of column values that are to be retrieved from the employee table. You must:

1. Declare an SQLDA structure and the variables that reference the SQLDA.
2. Dynamically allocate the SQLDA and the arrays needed for the column values.
3. Set the fields in the SQLDA for the column values to be retrieved.
4. Open the cursor.
5. Fetch the rows.

You must first declare the SQLDA structure. The following SQL INCLUDE statement requests a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA;
```

Your program must also declare variables that reference the SQLDA structure, the SQLVAR structure within the SQLDA, and the DECLEN structure for the precision and scale if you are retrieving a DECIMAL column. For C programs, the code looks like this:

```
struct sqlda *sqldaptr;
struct sqlvar *varptr;
struct DECLEN {
   unsigned char precision;
   unsigned char scale;
   };
```

Before you can set the fields in the SQLDA for the column values to be retrieved, you must dynamically allocate storage for the SQLDA structure. For C programs, the code looks like this:

```
sqldaptr = (struct sqlda *) malloc (3 * 44 + 16);
```

The size of the SQLDA is SQLN * 44 + 16, where the value of the SQLN field is the number of output columns.

You must set the fields in the SQLDA structure for your FETCH statement. Suppose you want to retrieve the columns EMPNO, LASTNAME, and SALARY. The C code to set the SQLDA fields for these columns looks like this:

```
strcpy(sqldaptr->sqldaid,"SQLDA");
sqldaptr->sqldabc = 148;      /* number bytes of storage allocated for the SQLDA */
sqldaptr->sqln = 3;                                   /* number of SQLVAR occurrences */
sqldaptr->sqld = 3;
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]));    /* Point to first SQLVAR */
varptr->sqltype = 452;                                       /* data type CHAR(6) */
varptr->sqllen = 6;
varptr->sqldata = (char *) hva1;
varptr->sqlind = (short *) inda1;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 1); /* Point to next SQLVAR */
varptr->sqltype = 448;                                   /* data type VARCHAR(15) */
varptr->sqllen = 15;
varptr->sqldata = (char *) hva2;
varptr->sqlind = (short *) inda2;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 2); /* Point to next SQLVAR */
varptr->sqltype = 485;                                  /* data type DECIMAL(9,2) */
((struct DECLEN *) &(varptr->sqllen))->precision = 9;
((struct DECLEN *) &(varptr->sqllen))->scale = 2;
varptr->sqldata = (char *) hva3;
varptr->sqlind = (short *) inda3;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
```

The SQLDA structure has these fields:

- SQLDABC indicates the number of bytes of storage that are allocated for the SQLDA. The storage includes a 16-byte header and 44 bytes for each SQLVAR field. The value is SQLN x 44 + 16, or 148 for this example.
- SQLN is the number of SQLVAR occurrences (or the number of output columns).
- SQLD is the number of variables in the SQLDA that are used by Db2 when processing the FETCH statement.
- Each SQLVAR occurrence describes a host-variable array or buffer into which the values for a column in the result table are to be returned. Within each SQLVAR:
  - SQLTYPE indicates the data type of the column.
  - SQLLEN indicates the length of the column. If the data type is DECIMAL, this field has two parts: the PRECISION and the SCALE.
  - SQLDATA points to the first element of the array for the column values. For this example, assume that your program allocates the dynamic variable arrays hva1, hva2, and hva3, and their indicator arrays inda1, inda2, and inda3.

- SQLIND points to the first element of the array of indicator values for the column. If SQLTYPE is an odd number, this attribute is required. (If SQLTYPE is an odd number, null values are allowed for the column.)
- SQLNAME has two parts: the LENGTH and the DATA. The LENGTH is 8. The first two bytes of the DATA field is X'0000'. Bytes 5 and 6 of the DATA field are a flag indicating whether the variable is an array or a FOR n ROWS value. Bytes 7 and 8 are a two-byte binary integer representation of the dimension of the array.

You can open the cursor only after all of the fields have been set in the output SQLDA:

```
EXEC SQL OPEN C1;
```

After the OPEN statement, the program fetches the next rowset:

```
EXEC SQL
  FETCH NEXT ROWSET FROM C1
  FOR 20 ROWS
  USING DESCRIPTOR :*sqldaptr;
```

The USING clause of the FETCH statement names the SQLDA that describes the columns that are to be retrieved.

After your program executes a FETCH statement to establish the current rowset, you can use a positioned UPDATE statement with either of the following clauses:

- Use WHERE CURRENT OF to modify all of the rows in the current rowset
- Use FOR ROW *n* OF ROWSET to modify row *n* in the current rowset

An example of a positioned UPDATE statement that uses the WHERE CURRENT OF clause is:

```
EXEC SQL
  UPDATE DSN8B10.EMP
    SET SALARY = 50000
    WHERE CURRENT OF C1
END-EXEC.
```

When the UPDATE statement is executed, the cursor must be positioned on a row or rowset of the result table. If the cursor is positioned on a row, that row is updated. If the cursor is positioned on a rowset, all of the rows in the rowset are updated.

An example of a positioned UPDATE statement that uses the FOR ROW *n* OF ROWSET clause is:

```
EXEC SQL
  UPDATE DSN8B10.EMP
    SET SALARY = 50000
    FOR CURSOR C1 FOR ROW 5 OF ROWSET
END-EXEC.
```

When the UPDATE statement is executed, the cursor must be positioned on a rowset of the result table. The specified row (in the example, row 5) of the current rowset is updated.

After your program executes a FETCH statement to establish the current rowset, you can use a positioned DELETE statement with either of the following clauses:

- Use WHERE CURRENT OF to delete all of the rows in the current rowset
- Use FOR ROW *n* OF ROWSET to delete row *n* in the current rowset

An example of a positioned DELETE statement that uses the WHERE CURRENT OF clause is:

```
EXEC SQL
  DELETE FROM DSN8B10.EMP
    WHERE CURRENT OF C1
END-EXEC.
```

When the DELETE statement is executed, the cursor must be positioned on a row or rowset of the result table. If the cursor is positioned on a row, that row is deleted, and the cursor is positioned before the next row of its result table. If the cursor is positioned on a rowset, all of the rows in the rowset are deleted, and the cursor is positioned before the next rowset of its result table.

An example of a positioned DELETE statement that uses the FOR ROW *n* OF ROWSET clause is:

```
EXEC SQL
  DELETE FROM DSN8B10.EMP
    FOR CURSOR C1 FOR ROW 5 OF ROWSET
END-EXEC.
```

When the DELETE statement is executed, the cursor must be positioned on a rowset of the result table. The specified row of the current rowset is deleted, and the cursor remains positioned on that rowset. The deleted row (in the example, row 5 of the rowset) cannot be retrieved or updated.

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Executing SQL statements by using a row cursor
You can use row cursors to execute FETCH statements, positioned UPDATE statements, and positioned DELETE statements.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Specifying the number of rows in a rowset
If you do not explicitly specify the number of rows in a rowset, Db2 implicitly determines the number of rows based on the last fetch request.

About this task

To explicitly set the size of a rowset, use the FOR *n* ROWS clause in the FETCH statement. If a FETCH statement specifies the ROWSET keyword, and not the FOR *n* ROWS clause, the size of the rowset is implicitly set to the size of the rowset that was most recently specified in a prior FETCH statement. If a prior FETCH statement did not specify the FOR *n* ROWS clause or the ROWSET keyword, the size of the current rowset is implicitly set to 1. For examples of rowset positioning, see Table 79 on page 426.

Closing a rowset cursor

Close a rowset cursor when it finishes processing rows if you want to free the resources or if you want to use the cursor again. Otherwise, you can let Db2 automatically close the cursor when the current transaction terminates or when your program terminates.

About this task

To free the resources held by the cursor, close the cursor explicitly by issuing the CLOSE statement.

If you want to use the rowset cursor again, reopen it.

Procedure

Issue a CLOSE statement.

Retrieving rows by using a scrollable cursor

A *scrollable cursor* is cursor that can be moved in both a forward and a backward direction. Scrollable cursors can be either row-positioned or rowset-positioned.

Procedure

When you open any cursor, the cursor is positioned before the first row of the result table. You move a scrollable cursor around in the result table by specifying a *fetch orientation* keyword in a FETCH statement.

A fetch orientation keyword indicates the absolute or relative position of the cursor when the FETCH statement is executed. The following table lists the fetch orientation keywords that you can specify and their meanings. These keywords apply to both row-positioned scrollable cursors and rowset-positioned scrollable cursors.

Table 77. Positions for a scrollable cursor

Keyword in FETCH statement	Cursor position when FETCH is executed"1.a" on page 415
BEFORE	Before the first row
FIRST or ABSOLUTE +1	On the first row
LAST or ABSOLUTE -1	On the last row
AFTER	After the last row
ABSOLUTE"1.b" on page 415	On an absolute row number, from before the first row forward or from after the last row backward
RELATIVE"1.b" on page 415	On the row that is forward or backward a relative number of rows from the current row
CURRENT	On the current row
PRIOR or RELATIVE -1	On the previous row
NEXT	On the next row (default)

Table notes

a. The cursor position applies to both row position and rowset position, for example, before the first row or before the first rowset.

b. For more information about ABSOLUTE and RELATIVE, see FETCH (Db2 SQL)

Example

To use the cursor that is declared in "Types of cursors" on page 401 to fetch the fifth row of the result table, use a FETCH statement like this:

```
EXEC SQL FETCH ABSOLUTE +5 C1 INTO :HVDEPTNO, :DEPTNAME, :MGRNO;
```

To fetch the fifth row from the end of the result table, use this FETCH statement:

```
EXEC SQL FETCH ABSOLUTE -5 C1 INTO :HVDEPTNO, :DEPTNAME, :MGRNO;
```

Related concepts

Types of cursors
You can declare row-positioned or rowset-positioned cursors in a number of ways. These cursors can be scrollable or not scrollable, held or not held, or returnable or not returnable.

Related reference

FETCH (Db2 SQL)

Comparison of scrollable cursors

Whether a scrollable cursor can view the changes that are made to the data by other processes or cursors depends on how the cursor is declared. It also depends on the type of fetch operation that is executed.

When you declare a cursor as SENSITIVE STATIC, changes that other processes or cursors make to the underlying table **can** be visible to the result table of the cursor. Whether those changes **are** visible depends on whether you specify SENSITIVE or INSENSITIVE when you execute FETCH statements with the cursor. When you specify FETCH INSENSITIVE, changes that other processes or other cursors make to the underlying table are not visible in the result table. When you specify FETCH SENSITIVE, changes that other processes or cursors make to the underlying table are visible in the result table.

When you declare a cursor as SENSITIVE DYNAMIC, changes that other processes or cursors make to the underlying table are visible to the result table after the changes are committed.

The following table summarizes the sensitivity values and their effects on the result table of a scrollable cursor.

Table 78. How sensitivity affects the result table for a scrollable cursor

DECLARE sensitivity	**FETCH INSENSITIVE**	**FETCH SENSITIVE**
INSENSITIVE	No changes to the underlying table are visible in the result table. Positioned UPDATE and DELETE statements using the cursor are not allowed.	Not valid.
SENSITIVE STATIC	Only positioned updates and deletes that are made by the cursor are visible in the result table.	All updates and deletes are visible in the result table. Inserts made by other processes are not visible in the result table.
SENSITIVE DYNAMIC	Not valid.	All committed changes are visible in the result table, including updates, deletes, inserts, and changes in the order of the rows.

Scrolling through a table in any direction

Use a scrollable cursor to move through the table in both a forward and a backward direction.

About this task

Question: How can I fetch rows from a table in any direction?

Answer: Declare your cursor as scrollable. When you select rows from the table, you can use the various forms of the FETCH statement to move to an absolute row number, move ahead or back a certain number of rows, to the first or last row, before the first row or after the last row, forward, or backward. You can use any combination of these FETCH statements to change direction repeatedly.

You can use code like the following example to move forward in the department table by 10 records, backward five records, and forward again by three records:

```
/**************************/
/* Declare host variables */
/**************************/
EXEC SQL BEGIN DECLARE SECTION;
  char[37] hv_deptname;
EXEC SQL END DECLARE SECTION;
/**********************************************************/
/* Declare scrollable cursor to retrieve department names */
/**********************************************************/
EXEC SQL DECLARE C1 SCROLL CURSOR FOR
  SELECT DEPTNAME FROM DSN8B10.DEPT;
⋮
/**********************************************************/
/* Open the cursor and position it before the start of    */
```

```
/* the result table.                                    */
/**********************************************************/
EXEC SQL OPEN C1;
EXEC SQL FETCH BEFORE FROM C1;
/**********************************************************/
/* Fetch first 10 rows                                    */
/**********************************************************/
for(i=0;i<10;i++)
{
  EXEC SQL FETCH NEXT FROM C1 INTO :hv_deptname;
}
/**********************************************************/
/* Save the value in the tenth row                        */
/**********************************************************/
tenth_row=hv_deptname;
/**********************************************************/
/* Fetch backward 5 rows                                  */
/**********************************************************/
for(i=0;i<5;i++)
{
  EXEC SQL FETCH PRIOR FROM C1 INTO :hv_deptname;
}
/**********************************************************/
/* Save the value in the fifth row                        */
/**********************************************************/
fifth_row=hv_deptname;
/**********************************************************/
/* Fetch forward 3 rows                                   */
/**********************************************************/
for(i=0;i<3;i++)
{
  EXEC SQL FETCH NEXT FROM C1 INTO :hv_deptname;
}
/**********************************************************/
/* Save the value in the eighth row                       */
/**********************************************************/
eighth_row=hv_deptname;
/**********************************************************/
/* Close the cursor                                       */
/**********************************************************/
EXEC SQL CLOSE C1;
```

Determining the number of rows in the result table for a static scrollable cursor

You can determine how many rows are in the result table of an INSENSITIVE or SENSITIVE STATIC scrollable cursor.

Procedure

To determine the number of rows in the result table for a static scrollable cursor, follow these steps:

1. Execute a FETCH statement, such as FETCH AFTER, that positions the cursor after the last row.
2. Perform one of the following actions:
   - Retrieve the values of fields SQLERRD(1) and SQLERRD(2) in the SQLCA (fields sqlerrd[0] and sqlerrd[1] for C and C++). SQLERRD(1) and SQLERRD(2) together form a double-word value that contains the number of rows in the result table.
   - Issue a GET DIAGNOSTICS statement to retrieve the value of the DB2_NUMBER_ROWS item.

Example

The following C language code demonstrates how to obtain the number of rows in a result table of a sensitive static cursor.

```
EXEC SQL INCLUDE SQLCA;
long int rowcount;
EXEC SQL
 DECLARE SENSTAT SENSITIVE STATIC SCROLL CURSOR FOR
 SELECT * FROM EMP;
EXEC SQL OPEN SENSTAT;
if (SQLCODE==0) {
 EXEC SQL FETCH AFTER SENSTAT; /* Position the cursor after the end */
                               /* of the result table               */
 if (SQLCODE==0) {
```

```
   /*************************************/
   /* Get the row count from the SQLCA */
   /*************************************/
   printf("%s \n","Row count from SQLCA: ");
   printf("%s %d\n","SQLERRD1: High-order word: ",sqlca.sqlerrd[0]);
                            /* Get the high-order word of the     */
                            /* result table size                  */
   printf("%s %d\n","SQLERRD2: Low-order word: ",sqlca.sqlerrd[1]);
                            /* Get the low-order word of the      */
                            /* result table size                  */
   /*******************************************/
   /* Get the row count from GET DIAGNOSTICS */
   /*******************************************/
   EXEC SQL GET DIAGNOSTICS :rowcount = DB2_NUMBER_ROWS;
   if (SQLCODE==0) {
    printf("%s %d\n","Row count from GET DIAGNOSTICS: ",rowcount);
   }
  }
 }
```

Removing a delete hole or update hole

If you try to fetch data from a delete hole or an update hole, Db2 issues an SQL warning. If you try to update or to delete a delete hole or delete an update hole, Db2 issues an SQL error.

About this task

You can remove a delete hole only by opening the scrollable cursor, setting a savepoint, executing a positioned DELETE statement with the scrollable cursor, and rolling back to the savepoint.

You can convert an update hole back to a result table row by updating the row in the base table, as shown in the following figure. You can update the base table with a searched UPDATE statement in the same application process, or a searched or positioned UPDATE statement in another application process. After you update the base table, if the row qualifies for the result table, the update hole disappears.

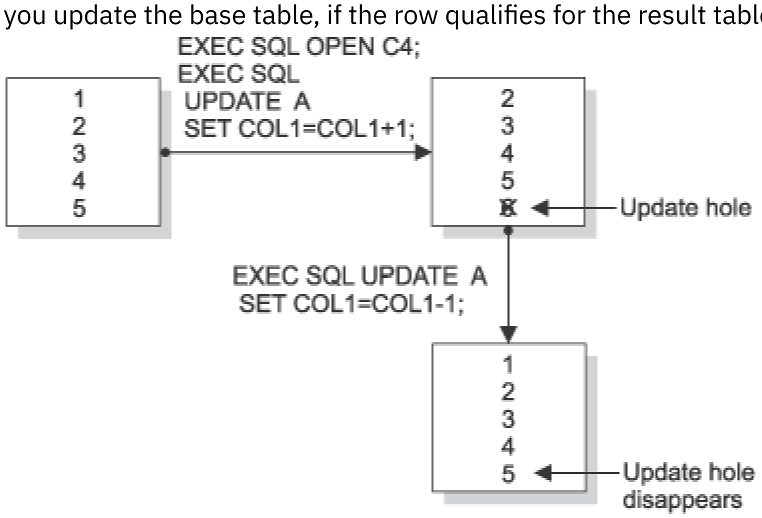


Figure 19. Removing an update hole

A hole becomes visible to a cursor when a cursor operation returns a non-zero SQLCODE. The point at which a hole becomes visible depends on the following factors:

- Whether the scrollable cursor creates the hole
- Whether the FETCH statement is FETCH SENSITIVE or FETCH INSENSITIVE

If the scrollable cursor creates the hole, the hole is visible when you execute a FETCH statement for the row that contains the hole. The FETCH statement can be FETCH INSENSITIVE or FETCH SENSITIVE.

If an update or delete operation outside the scrollable cursor creates the hole, the hole is visible at the following times:

- If you execute a FETCH SENSITIVE statement for the row that contains the hole, the hole is visible when you execute the FETCH statement.

- If you execute a FETCH INSENSITIVE statement, the hole is not visible when you execute the FETCH statement. Db2 returns the row as it was before the update or delete operation occurred. However, if you follow the FETCH INSENSITIVE statement with a positioned UPDATE or DELETE statement, the hole becomes visible.

Holes in the result table of a scrollable cursor
A hole in the result table means that the result table does not shrink to fill the space of deleted rows. It also does not shrink to fill the space of rows that have been updated and no longer satisfy the search condition. You cannot access a delete or update hole. However, you can remove holes in specific situations.

In some situations, you might not be able to fetch a row from the result table of a scrollable cursor, depending on how the cursor is declared:

- Scrollable cursors that are declared as INSENSITIVE or SENSITIVE STATIC follow a *static model*, which means that Db2 determines the size of the result table and the order of the rows when you open the cursor.

  Deleting or updating rows after a static cursor is open can result in holes in the result table. See "Removing a delete hole or update hole" on page 418.
- Scrollable cursors that are declared as SENSITIVE DYNAMIC follow a *dynamic model*, which means that the size and contents of the result table, and the order of the rows, can change after you open the cursor.

  A dynamic cursor scrolls directly on the base table. If the current row of the cursor is deleted or if it is updated so that it no longer satisfies the search condition, and the next cursor operation is FETCH CURRENT, then Db2 issues an SQL warning.

The following examples demonstrate how delete and update holes can occur when you use a SENSITIVE STATIC scrollable cursor.

Creating a delete hole with a static scrollable cursor:

Suppose that table A consists of one integer column, COL1, which has the values shown in the following figure.

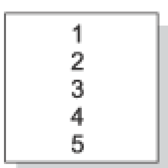


Figure 20. Values for COL1 of table A

Now suppose that you declare the following SENSITIVE STATIC scrollable cursor, which you use to delete rows from A:

```
EXEC SQL DECLARE C3 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT COL1
  FROM A
  FOR UPDATE OF COL1;
```

Now you execute the following SQL statements:

```
EXEC SQL OPEN C3;
EXEC SQL FETCH ABSOLUTE +3 C3 INTO :HVCOL1;
EXEC SQL DELETE FROM A WHERE CURRENT OF C3;
```

The positioned delete statement creates a delete hole, as shown in the following figure.

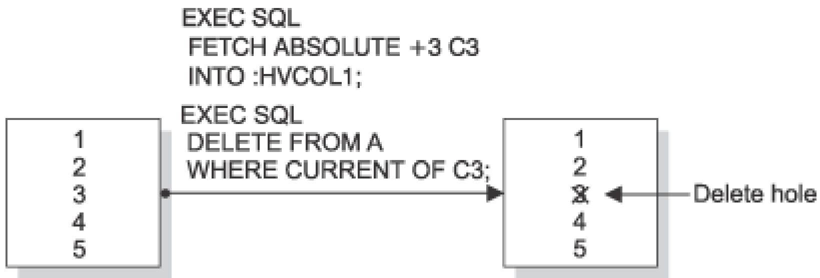


Figure 21. Delete hole

After you execute the positioned delete statement, the third row is deleted from the result table, but the result table does not shrink to fill the space that the deleted row creates.

Creating an update hole with a static scrollable cursor

Suppose that you declare the following SENSITIVE STATIC scrollable cursor, which you use to update rows in A:

```
EXEC SQL DECLARE C4 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT COL1
  FROM A
  WHERE COL1<6;
```

Now you execute the following SQL statements:

```
EXEC SQL OPEN C4;
UPDATE A SET COL1=COL1+1;
```

The searched UPDATE statement creates an update hole, as shown in the following figure.

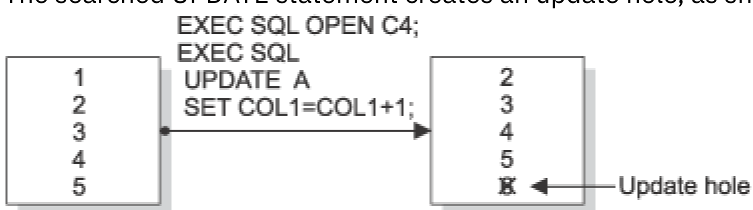


Figure 22. Update hole

After you execute the searched UPDATE statement, the last row no longer qualifies for the result table, but the result table does not shrink to fill the space that the disqualified row creates.

Accessing XML or LOB data quickly by using FETCH WITH CONTINUE

Use the FETCH WITH CONTINUE statement to improve the performance of some queries that reference XML and LOB columns with unknown or very large maximum lengths.

About this task

FETCH WITH CONTINUE breaks XML and LOB values into manageable pieces and processes the pieces one at a time to avoid the following buffer allocation problems:

- Allocating overly large or unnecessary space for buffers. If some LOB values are shorter than the maximum length for values in a column, you can waste buffer space if you allocate enough space for the maximum length. The buffer allocation problem can be even worse for XML data because an XML column does not have a defined maximum length. If you use FETCH WITH CONTINUE, you can allocate more appropriate buffer space for the actual length of the XML and LOB values.
- Truncating very large XML and LOB data. If a very large XML or LOB value does not fit in the host variable buffer space that is provided by the application program, Db2 truncates the value. If the application

program retries this fetch with a larger buffer, two problems exist. First, when using a non-scrollable cursor, you cannot re-fetch the current row without closing, reopening, and repositioning the cursor to the row that was truncated. Second, if you do not use FETCH WITH CONTINUE, Db2 does not return the actual length of the entire value to the application program. Thus, Db2 does not know how large a buffer to reallocate. If you use FETCH WITH CONTINUE, Db2 preserves the truncated portion of the data for subsequent retrieval and returns the actual length of the entire data value so that the application can reallocate a buffer of the appropriate size.

Db2 provides two methods for using FETCH WITH CONTINUE with LOB and XML data:

- “Dynamically allocating buffers when fetching XML and LOB data” on page 421
- “Moving data through fixed-size buffers when fetching XML and LOB data” on page 422

Dynamically allocating buffers when fetching XML and LOB data

If you specify FETCH WITH CONTINUE, Db2 returns information about which data does not fit in the buffer. Your application can then use the information about the truncated data to allocate an appropriate target buffer and execute a fetch operation with the CURRENT CONTINUE clause to retrieve the remaining data.

Procedure

To use dynamic buffer allocation for LOB and XML data:

1. Use an initial FETCH WITH CONTINUE to fetch data into a pre-allocated buffer of a moderate size.
2. If the value is too large to fit in the buffer, use the length information that is returned by Db2 to allocate the appropriate amount of storage.
3. Use a single FETCH CURRENT CONTINUE statement to retrieve the remainder of the data.

Example

Suppose that table T1 was created with the following statement:

```
CREATE TABLE T1 (C1 INT, C2 CLOB(100M), C3 CLOB(32K), C4 XML);
```

A row exists in T1 where C1 contains a valid integer, C2 contains 10MB of data, C3 contains 32KB of data, and C4 contains 4MB of data.

Now, suppose that you declare CURSOR1, prepare and describe statement DYNSQLSTMT1 with descriptor sqlda, and open CURSOR1 with the following statements:

```
EXEC SQL DECLARE CURSOR1 CURSOR FOR DYNSQLSTMT1;
EXEC SQL PREPARE DYNSQLSTMT1 FROM 'SELECT * FROM T1';
EXEC SQL DESCRIBE DYNSQLSTMT1 INTO DESCRIPTOR :SQLDA;
EXEC SQL OPEN CURSOR1;
```

Next, suppose that you allocate moderately sized buffers (32 KB for each CLOB or XML column) and set data pointers and lengths in SQLDA. Then, you use the following FETCH WITH CONTINUE statement:

```
EXEC SQL FETCH WITH CONTINUE CURSOR1 INTO DESCRIPTOR :SQLDA;
```

Because C2 and C4 contain data that do not fit in the buffer, some of the data is truncated. Your application can use the information that Db2 returns to allocate large enough buffers for the remaining data and reset the data pointers and length fields in SQLDA. At that point, you can resume the fetch and complete the process with the following FETCH CURRENT CONTINUE statement and CLOSE CURSOR statement:

```
EXEC SQL FETCH CURRENT CONTINUE CURSOR1 INTO DESCRIPTOR :SQLDA;
EXEC SQL CLOSE CURSOR1;
```

The application needs to concatenate the two returned pieces of the data value. One technique is to move the first piece of data to the dynamically-allocated larger buffer before the FETCH CONTINUE. Set the

SQLDATA pointer in the SQLDA structure to point immediately after the last byte of this truncated value. Db2 then writes the remaining data to this location and thus completes the concatenation.

Moving data through fixed-size buffers when fetching XML and LOB data

If you use the WITH CONTINUE clause, Db2 returns information about which data does not fit in the buffer. Your application can then use repeated FETCH CURRENT CONTINUE operations to effectively "stream" large XML and LOB data through a fixed-size buffer, one piece at a time.

Procedure

To use fixed buffer allocation for LOB and XML data, perform the following steps:

1. Use an initial FETCH WITH CONTINUE to fetch data into a pre-allocated buffer of a moderate size.
2. If the value is too large to fit in the buffer, use as many FETCH CONTINUE statements as necessary to process all of the data through a fixed buffer.

   After each FETCH operation, check whether a column was truncated by first examining the SQLWARN1 field in the returned SQLCA. If that field contains a 'W' value, at least one column in the returned row has been truncated. To then determine if a particular LOB or XML column was truncated, your application must compare the value that is returned in the length field with the declared length of the host variable. If a column is truncated, continue to use FETCH CONTINUE statements until all of the data has been retrieved.

   After you fetch each piece of the data, move it out of the buffer to make way for the next fetch. Your application can write the pieces to an output file or reconstruct the entire data value in a buffer above the 2-GB bar.

Example

Suppose that table T1 was created with the following statement:

```
CREATE TABLE T1 (C1 INT, C2 CLOB(100M), C3 CLOB(32K), C4 XML);
```

A row exists in T1 where C2 contains 10 MB of data.

Now, suppose that you declare a 32 KB section CLOBHV:

```
EXEC SQL BEGIN DECLARE SECTION
  DECLARE CLOBHV SQL TYPE IS CLOB(32767);
EXEC SQL END DECLARE SECTION.
```

Next, suppose that you use the following statements to declare and open CURSOR1 and to FETCH WITH CONTINUE:

```
EXEC SQL DECLARE CURSOR1 CURSOR FOR SELECT C2 FROM T1;
EXEC SQL OPEN CURSOR1;
EXEC SQL FETCH WITH CONTINUE CURSOR1 INTO :CLOBHV;
```

As each piece of the data value is fetched, move it from the buffer to the output file.

Because the 10 MB value in C2 does not fit into the 32 KB buffer, some of the data is truncated. Your application can loop through the following FETCH CURRENT CONTINUE:

```
EXEC SQL FETCH CURRENT CONTINUE CURSOR1 INTO :CLOBHV;
```

After each FETCH operation, you can determine if the data was truncated by first checking if the SQLWARN1 field in the returned SQLCA contains a 'W' value. If so, then check if the length value, which is returned in CLOBHV_LENGTH, is greater than the declared length of 32767. (CLOBHV_LENGTH is declared as part of the precompiler expansion of the CLOBHV declaration.) If the value is greater, that value has been truncated and more data can be retrieved with the next FETCH CONTINUE operation.

When all of the data has moved to the output file, you can close the cursor:

```
EXEC SQL CLOSE CURSOR1;
```

Determining the attributes of a cursor by using the SQLCA

An *SQL communications area (SQLCA)* is an area that is set apart for communication with Db2 and consists of a collection of variables. Using the SQLCA is one way to get information about any open cursors. Alternatively, you can use the GET DIAGNOSTICS statement.

About this task

After you open a cursor, you can determine the following attributes of the cursor by checking the following SQLWARN and SQLERRD fields of the SQLCA:

SQLWARN1
Indicates whether the cursor is scrollable or non-scrollable.

SQLWARN4
Indicates whether the cursor is insensitive (I), sensitive static (S), or sensitive dynamic (D).

SQLWARN5
Indicates whether the cursor is read-only, readable and deletable, or readable, deletable, and updatable.

SQLERRD(1) and SQLERRD(2)
These two fields together contain a double-word integer that represents the number of rows in the result table of a cursor when the cursor is positioned after the last row. The cursor is positioned after the last row when the SQLCODE is 100. These fields are not set for dynamic scrollable cursors.

SQLERRD(3)
The number of rows in the result table when the SELECT statement of the cursor contains a data change statement.

If the OPEN statement executes with no errors or warnings, Db2 does not set SQLWARN0 when it sets SQLWARN1, SQLWARN4, or SQLWARN5.

Related reference

Description of SQLCA fields (Db2 SQL)

Determining the attributes of a cursor by using the GET DIAGNOSTICS statement

Using the GET DIAGNOSTICS statement is one way to get information about any open cursors. Alternatively, you can use the SQLCA.

About this task

After you open a cursor, you can determine the following attributes of the cursor by checking these GET DIAGNOSTICS items:

DB2_SQL_ATTR_CURSOR_HOLD
Indicates whether the cursor can be held open across commits (Y or N)

DB2_SQL_ATTR_CURSOR_ROWSET
Indicates whether the cursor can use rowset positioning (Y or N)

DB2_SQL_ATTR_CURSOR_SCROLLABLE
Indicates whether the cursor is scrollable (Y or N)

DB2_SQL_ATTR_CURSOR_SENSITIVITY
Indicates whether the cursor is insensitive or sensitive to changes that are made by other processes (I or S)

DB2_SQL_ATTR_CURSOR_TYPE
Indicates whether the cursor is forward (F) declared static (S for INSENSITIVE or SENSITIVE STATIC) or dynamic (D for SENSITIVE DYNAMIC)

For more information about the GET DIAGNOSTICS statement, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

Scrolling through previously retrieved data

To scroll backward through data, use a scrollable cursor, or use a ROWID column or identity column to retrieve data in reverse order.

Procedure

When a program retrieves data from the database, it can scroll backward through the data by using one of the following techniques:

- Use a scrollable cursor to fetch backward through data by following these steps:
  a) Declare the cursor with the SCROLL keyword.
  b) Open the cursor.
  c) Execute a FETCH statement to position the cursor at the end of the result table.
  d) In a loop, execute FETCH statements that move the cursor backward and then retrieve the data.
  e) When you have retrieved all the data, close the cursor.

  For example, you can use code like the following example to retrieve department names in reverse order from table DSN8B10.DEPT:

```
/**************************/
/* Declare host variables */
/**************************/
EXEC SQL BEGIN DECLARE SECTION;
  char[37] hv_deptname;
EXEC SQL END DECLARE SECTION;
/*********************************************************/
/* Declare scrollable cursor to retrieve department names */
/*********************************************************/
EXEC SQL DECLARE C1 SCROLL CURSOR FOR
  SELECT DEPTNAME FROM DSN8B10.DEPT;
⋮
/*********************************************************/
/* Open the cursor and position it after the end of the   */
/* result table.                                          */
/*********************************************************/
EXEC SQL OPEN C1;
EXEC SQL FETCH AFTER FROM C1;
/*********************************************************/
/* Fetch rows backward until all rows are fetched.        */
/*********************************************************/
while(SQLCODE==0) {
  EXEC SQL FETCH PRIOR FROM C1 INTO :hv_deptname;
  ⋮
}
EXEC SQL CLOSE C1;
```

- If the table contains a ROWID or an identity column, retrieve the values from that column into an array. Then use the ROWID or identity column values to retrieve the rows in reverse order.

  You can use the ROWID column or identity column to rapidly retrieve the rows in reverse order. When you perform the original SELECT, you can store the ROWID or identity column value for each row you retrieve. Then, to retrieve the values in reverse order, you can execute SELECT statements with a WHERE clause that compares the ROWID or identity column value to each stored value.

  For example, suppose you add ROWID column DEPTROWID to table DSN8B10.DEPT. You can use code like the following example to select all department names, then retrieve the names in reverse order:

```
/**************************/
/* Declare host variables */
/**************************/
EXEC SQL BEGIN DECLARE SECTION;
```

```
  SQL TYPE IS ROWID  hv_dept_rowid;
  char[37] hv_deptname;
EXEC SQL END DECLARE SECTION;
/**************************/
/* Declare other variables */
/**************************/
struct rowid_struct {
  short int length;
  char data[40];    /* ROWID variable structure */
}
struct rowid_struct rowid_array[200];
                              /* Array to hold retrieved  */
                              /* ROWIDs.  Assume no more  */
                              /* than 200 rows will be    */
                              /* retrieved.               */
short int i,j,n;
/***********************************************/
/* Declare cursor to retrieve department names */
/***********************************************/
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNAME, DEPTROWID FROM DSN8B10.DEPT;
⋮
/**********************************************************/
/* Retrieve the department name and ROWID from DEPT table */
/* and store the ROWID in an array.                       */
/**********************************************************/
EXEC SQL OPEN C1;
i=0;
while(SQLCODE==0) {
  EXEC SQL FETCH C1 INTO :hv_deptname, :hv_dept_rowid;
  rowid_array[i].length=hv_dept_rowid.length;
  for(j=0;j<hv_dept_rowid.length;j++)
    rowid_array[i].data[j]=hv_dept_rowid.data[j];
  i++;
}
EXEC SQL CLOSE C1;
n=i-1;                /* Get the number of array elements */
/**********************************************************/
/* Use the ROWID values to retrieve the department names  */
/* in reverse order.                                      */
/**********************************************************/
for(i=n;i>=0;i--) {
  hv_dept_rowid.length=rowid_array[i].length;
  for(j=0;j<hv_dept_rowid.length;j++)
    hv_dept_rowid.data[j]=rowid_array[i].data[j];
  EXEC SQL SELECT DEPTNAME INTO :hv_deptname
    FROM DSN8B10.DEPT
    WHERE DEPTROWID=:hv_dept_rowid;
}
```

Related concepts

Row ID values (Db2 SQL)

Identity columns

An identity column contains a unique numeric value for each row in the table. Db2 can automatically generate sequential numeric values for this column as rows are inserted into the table. Thus, identity columns are ideal for primary key values, such as employee numbers or product numbers.

Related tasks

Retrieving rows by using a scrollable cursor

A *scrollable cursor* is cursor that can be moved in both a forward and a backward direction. Scrollable cursors can be either row-positioned or rowset-positioned.

Updating previously retrieved data

To scroll backward through data and update it, use a scrollable cursor that is declared with the FOR UPDATE clause.

About this task

Question: How can you scroll backward and update data that was retrieved previously?

Answer: Use a scrollable cursor that is declared with the FOR UPDATE clause.

Procedure

To update previously retrieved data:

1. Declare the cursor with the SENSITIVE STATIC SCROLL keywords.
2. Open the cursor.
3. Execute a FETCH statement to position the cursor at the end of the result table.
4. FETCH statements that move the cursor backward, until you reach the row that you want to update.
5. Execute the UPDATE WHERE CURRENT OF statement to update the current row.
6. Repeat steps "4" on page 426 and "5" on page 426 until you have update all the rows that you need to.
7. When you have retrieved and updated all the data, close the cursor.

FETCH statement interaction between row and rowset positioning

When you declare a cursor with the WITH ROWSET POSITIONING clause, you can intermix row-positioned FETCH statements with rowset-positioned FETCH statements.

The following table shows the interaction between row and rowset positioning for a scrollable cursor. Assume that you declare the scrollable cursor on a table with 15 rows.

Table 79. Interaction between row and rowset positioning for a scrollable cursor

Keywords in FETCH statement	Cursor position when FETCH is executed
FIRST	On row 1
FIRST ROWSET	On a rowset of size 1, consisting of row 1
FIRST ROWSET FOR 5 ROWS	On a rowset of size 5, consisting of rows 1, 2, 3, 4, and 5
CURRENT ROWSET	On a rowset of size 5, consisting of rows 1, 2, 3, 4, and 5
CURRENT	On row 1
NEXT (default)	On row 2
NEXT ROWSET	On a rowset of size 1, consisting of row 3
NEXT ROWSET FOR 3 ROWS	On a rowset of size 3, consisting of rows 4, 5, and 6
NEXT ROWSET	On a rowset of size 3, consisting of rows 7, 8, and 9
LAST	On row 15
LAST ROWSET FOR 2 ROWS	On a rowset of size 2, consisting of rows 14 and 15
PRIOR ROWSET	On a rowset of size 2, consisting of rows 12 and 13
ABSOLUTE 2	On row 2
ROWSET STARTING AT ABSOLUTE 2 FOR 3 ROWS	On a rowset of size 3, consisting of rows 2, 3, and 4
RELATIVE 2	On row 4
ROWSET STARTING AT ABSOLUTE 2 FOR 4 ROWS	On a rowset of size 4, consisting of rows 2, 3, 4, and 5
RELATIVE -1	On row 1
ROWSET STARTING AT ABSOLUTE 3 FOR 2 ROWS	On a rowset of size 2, consisting of rows 3 and 4
ROWSET STARTING AT RELATIVE 4	On a rowset of size 2, consisting of rows 7 and 8
PRIOR	On row 6

Keywords in FETCH statement	Cursor position when FETCH is executed
ROWSET STARTING AT ABSOLUTE 13 FOR 5 ROWS	On a rowset of size 3, consisting of rows 13, 14, and 15
FIRST ROWSET	On a rowset of size 5, consisting of rows 1, 2, 3, 4, and 5

Table 79. Interaction between row and rowset positioning for a scrollable cursor (continued)

Related reference

FETCH (Db2 SQL)

Examples of fetching rows by using cursors

You can use SQL statements that you include in a COBOL program to define and use non-scrollable cursor for row-positioned updates, scrollable cursors to retrieve rows backward, non-scrollable cursors for rowset-positioned updates, and scrollable cursors for rowset-positioned operations.

The following example shows how to update a row by using a cursor.

```
**************************************************
* Declare a cursor that will be used to update   *
* the JOB column of the EMP table.               *
**************************************************
 EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME,
      WORKDEPT, JOB
    FROM DSN8B10.EMP
    WHERE WORKDEPT = 'D11'
  FOR UPDATE OF JOB
 END-EXEC.
**************************************************
* Open the cursor                                *
**************************************************
 EXEC SQL
  OPEN THISEMP
 END-EXEC.
**************************************************
* Indicate what action to take when all rows     *
* in the result table have been fetched.         *
**************************************************
 EXEC SQL
  WHENEVER NOT FOUND
    GO TO CLOSE-THISEMP
 END-EXEC.
**************************************************
* Fetch a row to position the cursor.            *
**************************************************
 EXEC SQL
  FETCH FROM THISEMP
    INTO :EMP-NUM, :NAME2,
      :DEPT, :JOB-NAME
 END-EXEC.
**************************************************
* Update the row where the cursor is positioned. *
**************************************************
 EXEC SQL
  UPDATE DSN8B10.EMP
    SET JOB = :NEW-JOB
    WHERE CURRENT OF THISEMP
 END-EXEC.
⋮
**************************************************
* Branch back to fetch and process the next row. *
**************************************************
⋮
**************************************************
* Close the cursor                                *
**************************************************
 CLOSE-THISEMP.
  EXEC SQL
    CLOSE THISEMP
  END-EXEC.
```

The following example shows how to retrieve data backward with a cursor.

```
**************************************************
* Declare a cursor to retrieve the data backward *
* from the EMP table. The cursor has access to   *
* changes by other processes.                    *
**************************************************
 EXEC SQL
  DECLARE THISEMP SENSITIVE STATIC SCROLL CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8B10.EMP
 END-EXEC.
**************************************************
* Open the cursor                                *
**************************************************
 EXEC SQL
  OPEN THISEMP
 END-EXEC.
**************************************************
* Indicate what action to take when all rows     *
* in the result table have been fetched.         *
**************************************************
 EXEC SQL
  WHENEVER NOT FOUND GO TO CLOSE-THISEMP
 END-EXEC.
**************************************************
* Position the cursor after the last row of the  *
* result table. This FETCH statement cannot      *
* include the SENSITIVE or INSENSITIVE keyword   *
* and cannot contain an INTO clause.             *
**************************************************
 EXEC SQL
  FETCH AFTER FROM THISEMP
 END-EXEC.
**************************************************
* Fetch the previous row in the table.           *
**************************************************
 EXEC SQL
  FETCH SENSITIVE PRIOR FROM THISEMP
    INTO :EMP-NUM, :NAME2, :DEPT, :JOB-NAME
 END-EXEC.
**************************************************
* Check that the fetched row is not a hole       *
* (SQLCODE +222). If not, print the contents.    *
**************************************************
 IF SQLCODE IS GREATER THAN OR EQUAL TO 0 AND
    SQLCODE IS NOT EQUAL TO +100 AND
    SQLCODE IS NOT EQUAL TO +222 THEN
   PERFORM PRINT-RESULTS.
⋮
**************************************************
* Branch back to fetch the previous row.         *
**************************************************
⋮
**************************************************
* Close the cursor                               *
**************************************************
 CLOSE-THISEMP.
  EXEC SQL
    CLOSE THISEMP
  END-EXEC.
```

The following example shows how to update an entire rowset with a cursor.

```
**************************************************
* Declare a rowset cursor to update the JOB      *
* column of the EMP table.                       *
**************************************************
 EXEC SQL
  DECLARE EMPSET CURSOR
   WITH ROWSET POSITIONING FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8B10.EMP
    WHERE WORKDEPT = 'D11'
  FOR UPDATE OF JOB
 END-EXEC.
**************************************************
```

```
* Open the cursor.                              *
*************************************************
 EXEC SQL
  OPEN EMPSET
 END-EXEC.
*************************************************
* Indicate what action to take when end-of-data *
* occurs in the rowset being fetched.           *
*************************************************
 EXEC SQL
  WHENEVER NOT FOUND
    GO TO CLOSE-EMPSET
 END-EXEC.
*************************************************
* Fetch next rowset to position the cursor.     *
*************************************************
 EXEC SQL
  FETCH NEXT ROWSET FROM EMPSET
    FOR :SIZE-ROWSET ROWS
    INTO :HVA-EMPNO, :HVA-LASTNAME,
         :HVA-WORKDEPT, :HVA-JOB
 END-EXEC.
*************************************************
* Update rowset where the cursor is positioned. *
*************************************************
 UPDATE-ROWSET.
  EXEC SQL
    UPDATE DSN8B10.EMP
    SET JOB = :NEW-JOB
    WHERE CURRENT OF EMPSET
  END-EXEC.
 END-UPDATE-ROWSET.
⋮
*************************************************
* Branch back to fetch the next rowset.         *
*************************************************
⋮
*************************************************
* Update the remaining rows in the current      *
* rowset and close the cursor.                  *
*************************************************
 CLOSE-EMPSET.
  PERFORM UPDATE-ROWSET.
  EXEC SQL
    CLOSE EMPSET
  END-EXEC.
```

The following example shows how to update specific rows with a rowset cursor.

```
****************************************************
* Declare a static scrollable rowset cursor.       *
****************************************************
 EXEC SQL
   DECLARE EMPSET SENSITIVE STATIC SCROLL CURSOR
     WITH ROWSET POSITIONING FOR
     SELECT EMPNO, WORKDEPT, JOB
       FROM DSN8B10.EMP
       FOR UPDATE OF JOB
 END-EXEC.
****************************************************
* Open the cursor.                                 *
****************************************************
 EXEC SQL
   OPEN EMPSET
 END-EXEC.
****************************************************
* Fetch next rowset to position the cursor.        *
****************************************************
 EXEC SQL
   FETCH SENSITIVE NEXT ROWSET FROM EMPSET
     FOR :SIZE-ROWSET ROWS
     INTO :HVA-EMPNO,
          :HVA-WORKDEPT :INDA-WORKDEPT,
          :HVA-JOB :INDA-JOB
 END-EXEC.
****************************************************
* Process fetch results if no error and no hole.   *
****************************************************
```

```
 IF SQLCODE >= 0
   EXEC SQL GET DIAGNOSTICS
     :HV-ROWCNT = ROW_COUNT
   END-EXEC
   PERFORM VARYING N FROM 1 BY 1 UNTIL N > HV-ROWCNT
     IF INDA-WORKDEPT(N) NOT = -3
       EVALUATE HVA-WORKDEPT(N)
         WHEN ('D11')
           PERFORM UPDATE-ROW
         WHEN ('E11')
           PERFORM DELETE-ROW
       END-EVALUATE
     END-IF
   END-PERFORM
   IF SQLCODE = 100
     GO TO CLOSE-EMPSET
   END-IF
 ELSE
   EXEC SQL GET DIAGNOSTICS
     :HV-NUMCOND = NUMBER
   END-EXEC
   PERFORM VARYING N FROM 1 BY 1 UNTIL N > HV-NUMCOND
     EXEC SQL GET DIAGNOSTICS CONDITION :N
       :HV-SQLCODE = DB2_RETURNED_SQLCODE,
       :HV-ROWNUM = DB2_ROW_NUMBER
     END-EXEC
     DISPLAY "SQLCODE = " HV-SQLCODE
     DISPLAY "ROW NUMBER = " HV-ROWNUM
   END-PERFORM
   GO TO CLOSE-EMPSET
 END-IF.
```

```
⋮
*****************************************************
* Branch back to fetch and process                  *
* the next rowset.                                  *
*****************************************************
⋮
*****************************************************
* Update row N in current rowset.                   *
*****************************************************
 UPDATE-ROW.
   EXEC SQL
     UPDATE DSN8B10.EMP
     SET JOB = :NEW-JOB
     FOR CURSOR EMPSET FOR ROW :N OF ROWSET
   END-EXEC.
 END-UPDATE-ROW.
*****************************************************
* Delete row N in current rowset.                   *
*****************************************************
 DELETE-ROW.
   EXEC SQL
     DELETE FROM DSN8B10.EMP
     WHERE CURRENT OF EMPSET FOR ROW :N OF ROWSET
   END-EXEC.
 END-DELETE-ROW.
⋮
*****************************************************
* Close the cursor.                                 *
*****************************************************
 CLOSE-EMPSET.
  EXEC SQL
    CLOSE EMPSET
  END-EXEC.
```

Specifying direct row access by using row IDs

For some applications, you can use the value of a ROWID column to navigate directly to a row.

Before you begin

Ensure that the query qualifies for direct row access. To qualify, the search condition must be a Boolean term, stage 1 predicate that fits one of the following criteria:

- A simple Boolean term predicate of the following form:

```
RID (table designator) = noncolumn expression
```

Where the noncolumn expression contains a result of a RID function.

- A compound Boolean term that combines several simple predicates by using the AND operator, where one of the simple predicates fits the first criteria.

About this task

Introductory concepts

ROWID data type (Introduction to Db2 for z/OS)

A *ROWID column* uniquely identifies each row in a table. With ROWID columns you can write queries that navigate directly to a row in the table because the column implicitly contains the location of the row. You can define a ROWID column as either GENERATED BY DEFAULT or GENERATED ALWAYS:

- If you define the column as GENERATED BY DEFAULT, you can insert a value. Db2 provides a default value if you do not supply one. However, to be able to insert an explicit value (by using the INSERT statement with the VALUES clause), you must create a unique index on that column.
- If you define the column as GENERATED ALWAYS (which is the default), Db2 always generates a unique value for the column. You cannot insert data into that column. In this case, Db2 does not require an index to guarantee unique values.

When you select a ROWID column, the value implicitly contains the location of the retrieved row. If you use the value from the ROWID column in the search condition of a subsequent query, Db2 can choose to navigate directly to that row.

If you define a column in a table to have the ROWID data type, Db2 provides a unique value for each row in the table only if you define the column as GENERATED ALWAYS. The purpose of the value in the ROWID column is to uniquely identify rows in the table.

You can use a ROWID column to write queries that navigate directly to a row, which can be useful in situations where high performance is a requirement. This direct navigation, without using an index or scanning the table space, is called *direct row access*. In addition, a ROWID column is a requirement for tables that contain LOB columns. This topic discusses the use of a ROWID column in direct row access.

For example, suppose that an EMPLOYEE table is defined in the following way:

```
CREATE TABLE EMPLOYEE
  (EMP_ROWID   ROWID NOT NULL GENERATED ALWAYS,
   EMPNO       SMALLINT,
   NAME        CHAR(30),
   SALARY      DECIMAL(7,2),
   WORKDEPT    SMALLINT);
```

The following code uses the SELECT from INSERT statement to retrieve the value of the ROWID column from a new row that is inserted into the EMPLOYEE table. This value is then used to reference that row for the update of the SALARY column.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS ROWID hv_emp_rowid;
  short             hv_dept, hv_empno;
  char              hv_name[30];
  decimal(7,2)      hv_salary;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL
  SELECT EMP_ROWID INTO :hv_emp_rowid
  FROM FINAL TABLE (INSERT INTO EMPLOYEE
                  VALUES (DEFAULT, :hv_empno, :hv_name, :hv_salary, :hv_dept));
EXEC SQL
  UPDATE EMPLOYEE
  SET SALARY = SALARY + 1200
  WHERE EMP_ROWID = :hv_emp_rowid;

EXEC SQL COMMIT;
```

For Db2 to be able to use direct row access for the update operation, the SELECT from INSERT statement and the UPDATE statement must execute within the same unit of work. Alternatively, you can use a SELECT from MERGE statement. The MERGE statement performs INSERT and UPDATE operations as one coordinated statement.

Requirement: To use direct row access, you must use a retrieved ROWID value before you commit. When your application commits, it releases its claim on the table space. After the commit, if a REORG is run on your table space, the physical location of the rows might change.

Restriction: In general, you cannot use a ROWID column as a key that is to be used as a single column value across multiple tables. The ROWID value for a particular row in a table might change over time due to a REORG of the table space. In particular, you cannot use a ROWID column as part of a parent key or foreign key.

The value that you retrieve from a ROWID column is a varying-length character value that is not monotonically ascending or descending (the value is not always increasing or not always decreasing). Therefore, a ROWID column does not provide suitable values for many types of entity keys, such as order numbers or employee numbers.

Procedure

Call the RID built-in function in the search condition of a SELECT, DELETE, or UPDATE statement.

The RID function returns the RID of a row, which you can use to uniquely identify a row.

Restriction: Because Db2 might reuse RID numbers when the REORG utility is run, the RID function might return different values when invoked for a row multiple times.

If you specify a RID and Db2 cannot locate the row through direct row access, Db2 does not switch to another access method. Instead, Db2 returns no rows.

Related concepts

Rules for inserting data into a ROWID column
A *ROWID column* contains unique values that identify each row in a table. Whether you can insert data into a ROWID column and how that data gets inserted depends on how the column is defined.

Direct row access (PRIMARY_ACCESSTYPE='D') (Db2 Performance)

Row ID values (Db2 SQL)

Related reference

RID (Db2 SQL)

Ways to manipulate LOB data

You can use SQL statements, LOB locators, and LOB file reference variables in your application programs to manipulate LOB data that is stored in Db2.

For example, you can use the following statements to extract information about an employee's department from the resume:

```
EXEC SQL BEGIN DECLARE SECTION;
  char     employeenum[6];
  long     deptInfoBeginLoc;
  long     deptInfoEndLoc;
  SQL TYPE IS CLOB_LOCATOR resume;
  SQL TYPE IS CLOB_LOCATOR deptBuffer;
EXEC SQL END DECLARE SECTION;
⋮
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT EMPNO, EMP_RESUME FROM EMP;
⋮
EXEC SQL FETCH C1 INTO :employeenum, :resume;
⋮
EXEC SQL SET  :deptInfoBeginLoc =
  POSSTR(:resume.data, 'Department Information');

EXEC SQL SET :deptInfoEndLoc =
  POSSTR(:resume.data, 'Education');
```

```
EXEC SQL SET :deptBuffer =
  SUBSTR(:resume, :deptInfoBeginLoc,
  :deptInfoEndLoc - :deptInfoBeginLoc);
```

These statements use host variables of data type large object locator (LOB locator). LOB locators let you manipulate LOB data without moving the LOB data into host variables. By using LOB locators, you need much smaller amounts of memory for your programs.

You can also use LOB file reference variables when you are working with LOB data. You can use LOB file reference variables to insert LOB data from a file into a Db2 table or to retrieve LOB data from a Db2 table.

Sample LOB applications: The following table lists the sample programs that Db2 provides to assist you in writing applications to manipulate LOB data. All programs reside in data set DSN1110.SDSNSAMP.

Table 80. LOB samples shipped with Db2

Member that contains source code	Language	Function
DSNTEJ7	JCL	Demonstrates how to create a table with LOB columns, an auxiliary table, and an auxiliary index. Also demonstrates how to load LOB data that is 32 KB or less into a LOB table space.
DSN8DLPL	C	Demonstrates the use of LOB locators and UPDATE statements to move binary data into a column of type BLOB.
DSN8DLRV	C	Demonstrates how to use a locator to manipulate data of type CLOB.
DSNTEP2	PL/I	Demonstrates how to allocate an SQLDA for rows that include LOB data and use that SQLDA to describe an input statement and fetch data from LOB columns.

Related concepts

LOB file reference variables
In a host application, you can use a file reference variable to insert a LOB or XML value from a file into a Db2 table. You can also use a file reference variable to select a LOB or XML value from a Db2 table into a file.

Phase 7: Accessing LOB data (Db2 Installation and Migration)

Related tasks

Saving storage when manipulating LOBs by using LOB locators
LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. By using locators, you avoid needing to allocate the large amounts of storage that are needed for host variables to hold LOB data.

LOB host variable, LOB locator, and LOB file reference variable declarations

When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

You can declare LOB host variables and LOB locators in assembler, C, C++, COBOL, Fortran, and PL/I. Additionally, you can declare LOB file reference variables in assembler, C, C++, COBOL, and PL/I. For each host variable, locator, or file reference variable of SQL type BLOB, CLOB, or DBCLOB that you declare, Db2 generates an equivalent declaration that uses host language data types. When you refer to a LOB host variable, LOB locator, or LOB file reference variable in an SQL statement, you must use the variable that you specified in the SQL type declaration. When you refer to the host variable in a host language statement, you must use the variable that Db2 generates.

Db2 supports host variable declarations for LOBs with lengths of up to 2 GB - 1. However, the size of a LOB host variable is limited by the restrictions of the host language and the amount of storage available to the program.

Declare LOB host variables that are referenced by the precompiler in SQL statements by using the SQL TYPE IS BLOB, SQL TYPE IS CLOB, or SQL TYPE IS DBCLOB keywords.

LOB host variables that are referenced only by an SQL statement that uses a DESCRIPTOR should use the same form as declared by the precompiler. In this form, the LOB host-variable-array consists of a 31-bit length, followed by the data, followed by another 31-bit length, followed by the data, and so on. The 31-bit length must be fullword aligned.

Example: Suppose that you want to allocate a LOB array of 10 elements, each with a length of 5 bytes. You need to allocate the following bytes for each element, for a total of 120 bytes:

- 4 bytes for the 31-bit integer
- 5 bytes for the data
- 3 bytes to force fullword alignment

The following examples show you how to declare LOB host variables in each supported language. In each table, the left column contains the declaration that you code in your application program. The right column contains the declaration that Db2 generates.

Declarations of LOB host variables in assembler

The following table shows assembler language declarations for some typical LOB types.

Table 81. Example of assembler LOB variable declarations

You declare this variable	Db2 generates this variable

`clob_var SQL TYPE IS CLOB 40000K`	`clob_var DS 0FL4`
	`clob_var_length DS FL4`
	`clob_var_data DS CL65535`[1]
	`ORG clob_var_data +(40960000-65535)`

`dbclob_var SQL TYPE IS DBCLOB 4000K`	`dbclob_var DS 0FL4`
	`dbclob_var_length DS FL4`
	`dbclob_var_data DS GL65534`[2]
	`ORG dbclob_var_data+(8192000-65534)`

`blob_var SQL TYPE IS BLOB 1M`	`blob_var DS 0FL4`
	`blob_var_length DS FL4`
	`blob_var_data DS CL65535`[1]
	`ORG blob_var_data+(1048476-65535)`

`clob_loc SQL TYPE IS CLOB_LOCATOR`	`clob_loc DS FL4`
`dbclob_loc SQL TYPE IS DBCLOB_LOCATOR`	`dbclob_loc DS FL4`
`blob_loc SQL TYPE IS BLOB_LOCATOR`	`blob_loc DS FL4`
`clob_file SQL TYPE IS CLOB_FILE`	`clob_file DS FL4`
`dbclob_file SQL TYPE IS DBCLOB_FILE`	`dbclob_file DS FL4`
`blob_file SQL TYPE IS BLOB_FILE`	`blob_file DS FL4`

Table 81. Example of assembler LOB variable declarations (continued)	
You declare this variable	**Db2 generates this variable**

Notes:

1. Because assembler language allows character declarations of no more than 65535 bytes, Db2 separates the host language declarations for BLOB and CLOB host variables that are longer than 65535 bytes into two parts.
2. Because assembler language allows graphic declarations of no more than 65534 bytes, Db2 separates the host language declarations for DBCLOB host variables that are longer than 65534 bytes into two parts.

Declarations of LOB host variables in C and C++

The following table shows C and C++ language declarations for some typical LOB types.

Table 82. Examples of C language variable declarations

You declare this variable	Db2 generates this variable

`SQL TYPE IS BLOB (1M) blob_var;`	`struct {`
	`   unsigned long length;`
	`   char data[1048576];`
	`} blob_var;`

`SQL TYPE IS CLOB(400K) clob_var;`	`struct {`
	`   unsigned long length;`
	`   char data[409600];`
	`} clob_var;`

`SQL TYPE IS DBCLOB (4000K) dbclob_var;`	`struct {`
	`   unsigned long length;`
	`   sqldbchar data[4096000];`
	`} dbclob_var;`

`SQL TYPE IS BLOB_LOCATOR blob_loc;`	`unsigned long blob_loc;`
`SQL TYPE IS CLOB_LOCATOR clob_loc;`	`unsigned long clob_loc;`
`SQL TYPE IS DBCLOB_LOCATOR dbclob_loc;`	`unsigned long dbclob_loc;`

`SQL TYPE IS BLOB_FILE FBLOBhv;`	`#pragma pack(full)`
	`struct {`
	`unsigned long name_length;`
	`unsigned long data_length;`
	`unsigned long file_options;`
	`char name??(255??);`
	`} FBLOBhv        ;`
	`#pragma pack(reset)`

`SQL TYPE IS CLOB_FILE FCLOBhv;`	`#pragma pack(full)`
	`struct {`
	`unsigned long name_length;`
	`unsigned long data_length;`
	`unsigned long file_options;`
	`char name??(255??);`
	`} FCLOBhv        ;`
	`#pragma pack(reset)`

Table 82. Examples of C language variable declarations (continued)

You declare this variable	Db2 generates this variable

`SQL TYPE IS DBCLOB_FILE FDBCLOBhv;`	`#pragma pack(full)`
	`struct {`
	`unsigned long name_length;`
	`unsigned long data_length;`
	`unsigned long file_options;`
	`char name??(255??);`
	`} FDBCLOBhv        ;`
	`#pragma pack(reset)`

Declarations of LOB host variables in COBOL

The declarations that are generated for COBOL depend on whether you use the Db2 precompiler or the Db2 coprocessor. The following table shows COBOL declarations that the Db2 precompiler generates for some typical LOB types. The declarations that the Db2 coprocessor generates might be different.

Table 83. Examples of COBOL variable declarations by the Db2 precompiler

You declare this variable	Db2 precompiler generates this variable

`01  BLOB-VAR`
`    SQL TYPE IS BLOB(1M).`
	`01  BLOB-VAR.`
	`    49 BLOB-VAR-LENGTH PIC S9(9) COMP-5.`
	`    49 BLOB-VAR-DATA PIC X(1048576).`

`01  CLOB-VAR`
`    SQL TYPE IS CLOB(40000K).`
	`01  CLOB-VAR.`
	`    49 CLOB-VAR-LENGTH PIC S9(9) COMP-5.`
	`    49 CLOB-VAR-DATA PIC X(40960000).`

`01  DBCLOB-VAR`
`    SQL TYPE IS DBCLOB(4000K).`
	`01  DBCLOB-VAR.`
	`    49 DBCLOB-VAR-LENGTH PIC S9(9) COMP-5`
	`    49 DBCLOB-VAR-DATA PIC G(40960000)`
	`    DISPLAY-1.`

`01  BLOB-LOC` `    SQL TYPE IS BLOB-LOCATOR.`	`01  BLOB-LOC PIC S9(9) COMP-5.`
`01  CLOB-LOC` `    SQLTYPE IS CLOB-LOCATOR.`	`01  CLOB-LOC PIC S9(9) COMP-5.`
`01  DBCLOB-LOC` `    SQLTYPE IS DBCLOB-LOCATOR.`	`01  DBCLOB-LOC PIC S9(9) COMP-5.`

`01  BLOB-FILE`
`    SQLTYPE IS BLOB-FILE.`
	`01  BLOB-FILE.`
	`  49 BLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.`
	`  49 BLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.`
	`  49 BLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.`
	`  49 BLOB-FILE-NAME PIC X(255) .`

`01  CLOB-FILE`
`    SQLTYPE IS CLOB-FILE.`
	`01  CLOB-FILE.`
	`  49 CLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.`
	`  49 CLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.`
	`  49 CLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.`
	`  49 CLOB-FILE-NAME PIC X(255) .`

Table 83. Examples of COBOL variable declarations by the Db2 precompiler (continued)

You declare this variable	Db2 precompiler generates this variable

01 DBCLOB-FILE
    SQLTYPE IS DBCLOB-FILE.
	01 DBCLOB-FILE.
	  49 DBCLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.
	  49 DBCLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5.
	  49 DBCLOB-FILE-FILE-OPTION PIC S9(9) COMP-5.
	  49 DBCLOB-FILE-NAME PIC X(255) .

Declarations of LOB host variables in Fortran

The following table shows Fortran declarations for some typical LOB types.

Table 84. Examples of Fortran variable declarations

You declare this variable	Db2 generates this variable
SQL TYPE IS BLOB(1M) blob_var	CHARACTER blob_var(1048580) INTEGER*4 blob_var_LENGTH CHARACTER blob_var_DATA EQUIVALENCE(blob_var(1), + blob_var_LENGTH) EQUIVALENCE(blob_var(5), + blob_var_DATA)
SQL TYPE IS CLOB(40000K) clob_var	CHARACTER clob_var(4096004) INTEGER*4 clob_var_length CHARACTER clob_var_data EQUIVALENCE(clob_var(1), + clob_var_length) EQUIVALENCE(clob_var(5), + clob_var_data)
SQL TYPE IS BLOB_LOCATOR blob_loc	INTEGER*4 blob_loc
SQL TYPE IS CLOB_LOCATOR clob_loc	INTEGER*4 clob_loc

Declarations of LOB host variables in PL/I

The declarations that are generated for PL/I depend on whether you use the Db2 precompiler or the Db2 coprocessor. The following table shows PL/I declarations that the Db2 precompiler generates for some typical LOB types. The declarations that the Db2 coprocessor generates might be different.

Table 85. Examples of PL/I variable declarations by the Db2 precompiler

You declare this variable	Db2 precompiler generates this variable

DCL BLOB_VAR
  SQL TYPE IS BLOB (1M);
	DCL 1 BLOB_VAR,
	  2 BLOB_VAR_LENGTH FIXED BINARY(31),
	  2 BLOB_VAR_DATA,[1]
	    3 BLOB_VAR_DATA1(32)
	        CHARACTER(32767),
	    3 BLOB_VAR_DATA2
	        CHARACTER(1048576-32*32767);

Table 85. Examples of PL/I variable declarations by the Db2 precompiler (continued)

You declare this variable	Db2 precompiler generates this variable
DCL CLOB_VAR SQL TYPE IS CLOB (40000K);	DCL 1 CLOB_VAR, 2 CLOB_VAR_LENGTH FIXED BINARY(31), 2 CLOB_VAR_DATA,[1] 3 CLOB_VAR_DATA1(1250) CHARACTER(32767), 3 CLOB_VAR_DATA2 CHARACTER(40960000-1250*32767);
DCL DBCLOB_VAR SQL TYPE IS DBCLOB (4000K);	DCL 1 DBCLOB_VAR, 2 DBCLOB_VAR_LENGTH FIXED BINARY(31), 2 DBCLOB_VAR_DATA,[2] 3 DBCLOB_VAR_DATA1(250) GRAPHIC(16383), 3 DBCLOB_VAR_DATA2 GRAPHIC(4096000-250*16383);
DCL blob_loc SQL TYPE IS BLOB_LOCATOR;	DCL blob_loc FIXED BINARY(31);
DCL clob_loc SQL TYPE IS CLOB_LOCATOR;	DCL clob_loc FIXED BINARY(31);
DCL dbclob_loc SQL TYPE IS DBCLOB_LOCATOR;	DCL dbclob_loc FIXED BINARY(31);
DCL blob_file SQL TYPE IS BLOB_FILE;	DCL 1 blob_file, 2 blob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 blob_file_DATA_LENGTH BIN FIXED(31), 2 blob_file_FILE_OPTIONS BIN FIXED(31), 2 blob_file_NAME CHAR(255) ;
DCL clob_file SQL TYPE IS CLOB_FILE;	DCL 1 clob_file, 2 clob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 clob_file_DATA_LENGTH BIN FIXED(31), 2 clob_file_FILE_OPTIONS BIN FIXED(31), 2 clob_file_NAME CHAR(255) ;
DCL dbclob_file SQL TYPE IS DBCLOB_FILE;	DCL 1 dbclob_file, 2 dbclob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 dbclob_file_DATA_LENGTH BIN FIXED(31), 2 dbclob_file_FILE_OPTIONS BIN FIXED(31), 2 dbclob_file_NAME CHAR(255) ;

Table 85. Examples of PL/I variable declarations by the Db2 precompiler (continued)

You declare this variable	Db2 precompiler generates this variable

Notes:

1. For BLOB or CLOB host variables that are greater than 32767 bytes in length, Db2 creates PL/I host language declarations in the following way:
   - If the length of the LOB is greater than 32767 bytes and evenly divisible by 32767, Db2 creates an array of 32767-byte strings. The dimension of the array is *length*/32767.
   - If the length of the LOB is greater than 32767 bytes but not evenly divisible by 32767, Db2 creates two declarations: The first is an array of 32767 byte strings, where the dimension of the array, *n*, is *length*/32767. The second is a character string of length *length*-*n**32767.
2. For DBCLOB host variables that are greater than 16383 double-byte characters in length, Db2 creates PL/I host language declarations in the following way:
   - If the length of the LOB is greater than 16383 characters and evenly divisible by 16383, Db2 creates an array of 16383-character strings. The dimension of the array is *length*/16383.
   - If the length of the LOB is greater than 16383 characters but not evenly divisible by 16383, Db2 creates two declarations: The first is an array of 16383 byte strings, where the dimension of the array, *m*, is *length*/16383. The second is a character string of length *length*-*m**16383.

Related concepts

LOB file reference variables
In a host application, you can use a file reference variable to insert a LOB or XML value from a file into a Db2 table. You can also use a file reference variable to select a LOB or XML value from a Db2 table into a file.

Related tasks

Saving storage when manipulating LOBs by using LOB locators
LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. By using locators, you avoid needing to allocate the large amounts of storage that are needed for host variables to hold LOB data.

LOB and XML materialization

Materialization means that Db2 puts the data that is selected into a buffer for processing. This action can slow performance. Because LOB values can be very large, Db2 avoids materializing LOB data until absolutely necessary.

Beginning in DB2 10, LOB and XML materialization has been reduced or eliminated within Db2 for several local and distributed cases including utilities (LOAD and cross-loader). Some of the cases where materialization has been eliminated or reduced include during DRDA streaming, file reference variable processing, CCSID conversion and distributed XML fetch processing. However, whether the values will be materialized and how much will be materialized also depends on the number and size of each LOB or XML.

Db2 stores LOB values in contiguous storage. Db2 must materialize LOBs when your application program performs the following actions:

- Calls a user-defined function with a LOB as an argument
- Moves a LOB into or out of a stored procedure
- Assigns a LOB host variable to a LOB locator host variable

The amount of storage that is used for LOB and XML materialization depends on a number of factors including:

- The size of the LOBs
- The number of LOBs that need to be materialized in a statement

Db2 loads LOBs into virtual pools above the bar. If insufficient space is available for LOB materialization, your application receives SQLCODE -904.

Although you cannot completely avoid LOB materialization, you can minimize it by using LOB locators, rather than LOB host variables in your application programs.

Related tasks

Saving storage when manipulating LOBs by using LOB locators
LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. By using locators, you avoid needing to allocate the large amounts of storage that are needed for host variables to hold LOB data.

Saving storage when manipulating LOBs by using LOB locators

LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. By using locators, you avoid needing to allocate the large amounts of storage that are needed for host variables to hold LOB data.

About this task

To retrieve LOB data from a Db2 table, you can define host variables that are large enough to hold all of the LOB data. This requires your application to allocate large amounts of storage, and requires Db2 to move large amounts of data, which can be inefficient or impractical. Instead, you can use LOB locators. LOB locators let you manipulate LOB data without retrieving the data from the Db2 table. Using LOB locators for LOB data retrieval is a good choice in the following situations:

- When you move only a small part of a LOB to a client program
- When the entire LOB does not fit in the application's memory
- When the program needs a temporary LOB value from a LOB expression but does not need to save the result
- When performance is important

A LOB locator is associated with a LOB value or expression, not with a row in a Db2 table or a physical storage location in a table space. Therefore, after you select a LOB value using a locator, the value in the locator normally does not change until the current unit of work ends. However the value of the LOB itself can change.

If you want to remove the association between a LOB locator and its value before a unit of work ends, execute the FREE LOCATOR statement. To keep the association between a LOB locator and its value after the unit of work ends, execute the HOLD LOCATOR statement. After you execute a HOLD LOCATOR statement, the locator keeps the association with the corresponding value until you execute a FREE LOCATOR statement or the program ends.

If you execute HOLD LOCATOR or FREE LOCATOR dynamically, you cannot use EXECUTE IMMEDIATE.

Applications that use a huge number of locators, which commit infrequently, or do not explicitly free the locators, can use large amounts of valuable database services address space (*ssnm*DBM1) storage and CPU costs. Frequently use COMMIT or FREE LOCATORS to avoid storage shortage on the database services address space (*ssnm*DBM1) and a shortage of system CPU resource.

To free LOB locators after their associated LOB values are retrieved, run the FREE LOCATOR statement:

```
EXEC SQL FREE LOCATOR :LOCRES, :LOCHIST, :LOCPIC
```

Related reference

FREE LOCATOR (Db2 SQL)

HOLD LOCATOR (Db2 SQL)

Indicator variables and LOB locators

Db2 uses indicator variables for LOB locators differently than it uses indicator variables for host variables.

For host variables other than LOB locators, when you select a null value into a host variable, Db2 assigns a negative value to the associated indicator variable. However, for LOB locators, Db2 uses indicator variables differently. A LOB locator is never null. When you select a LOB column using a LOB locator and the LOB column contains a null value, Db2 assigns a null value to the associated indicator variable. The value in the LOB locator does not change. In a client/server environment, this null information is recorded only at the client.

When you use LOB locators to retrieve data from columns that can contain null values, define indicator variables for the LOB locators, and check the indicator variables after you fetch data into the LOB locators. If an indicator variable is null after a fetch operation, you cannot use the value in the LOB locator.

Valid assignments for LOB locators

Although you usually use LOB locators to assign data to and retrieve data from LOB columns, you can also use LOB locators to assign data to non-LOB columns.

You can use LOB locators to make the following assignments:

- A CLOB or DBCLOB locator can be assigned to a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC column. However, you cannot fetch data from CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC columns into a CLOB or DBCLOB locators.
- A BLOB locator can be assigned to a BINARY or VARBINARY column. However, you cannot fetch data from a BINARY or VARBINARY column into a BLOB locator.

Avoiding character conversion for LOB locators

In certain situations, Db2 materializes the entire LOB value and converts it to the encoding scheme of a particular SQL statement. This extra processing can degrade performance and should be avoided.

About this task

You can use a VALUES INTO or SET statement to obtain the results of functions that operate on LOB locators, such as LENGTH or SUBSTR. VALUES INTO and SET statements are processed in the application encoding scheme for the plan or package that contains the statement. If that encoding scheme is different from the encoding scheme of the LOB data, the entire LOB value is materialized and converted to the encoding scheme of the statement. This materialization and conversion processing can cause performance degradation.

To avoid the character conversion, SELECT from the SYSIBM.SYSDUMMYA, SYSIBM.SYSDUMMYE, or SYSIBM.SYSDUMMYU sample table. These dummy tables perform functions similar to SYSIBM.SYSDUMMY1, and are each associated with an encoding scheme:

SYSIBM.SYSDUMMYA
: ASCII

SYSIBM.SYSDUMMYE
: EBCDIC

SYSIBM.SYSDUMMYU
: Unicode

By using these tables, you can obtain the same result as you would with a VALUES INTO or SET statement.

Example

Suppose that the encoding scheme of the following statement is EBCDIC:

```
SET : unicode_hv = SUBSTR(:Unicode_lob_locator,X,Y);
```

Db2 must materialize the LOB that is specified by :Unicode_lob_locator and convert that entire LOB to EBCDIC before executing the statement. To avoid materialization and conversion, you can execute the following statement, which produces the same result but is processed by the Unicode encoding scheme of the table:

```
SELECT SUBSTR(:Unicode_lob_locator,X,Y) INTO :unicode_hv
    FROM SYSIBM.SYSDUMMYU;
```

Deferring evaluation of a LOB expression to improve performance

Db2 does not move any bytes of a LOB value until a program assigns a LOB expression to a target destination. When you use a LOB locator with string functions and operators, Db2 does not evaluate the expression until the time of assignment. This deferred evaluation can improve performance.

About this task

The following example is a C language program that defers evaluation of a LOB expression. The program runs on a client and modifies LOB data at a server. The program searches for a particular resume (EMPNO = '000130') in the EMP_RESUME table. It then uses LOB locators to rearrange a copy of the resume (with EMPNO = 'A00130'). In the copy, the Department Information Section appears at the end of the resume. The program then inserts the copy into EMP_RESUME without modifying the original resume.

Because the program in the following figure uses LOB locators, rather than placing the LOB data into host variables, no LOB data is moved until the INSERT statement executes. In addition, no LOB data moves between the client and the server.

```
EXEC SQL INCLUDE SQLCA;

/**************************/
/* Declare host variables */                           1
/**************************/
EXEC SQL BEGIN DECLARE SECTION;
   char userid[9];
   char passwd[19];
   long            HV_START_DEPTINFO;
   long            HV_START_EDUC;
   long            HV_RETURN_CODE;
   SQL TYPE IS CLOB_LOCATOR HV_NEW_SECTION_LOCATOR;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR1;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR2;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR3;
EXEC SQL END DECLARE SECTION;

/*************************************************/
/* Delete any instance of "A00130" from previous */
/* executions of this sample                     */
/*************************************************/
EXEC SQL DELETE FROM EMP_RESUME WHERE EMPNO = 'A00130';

/*************************************************/
/* Use a single row select to get the document   */       2
/*************************************************/
EXEC SQL SELECT RESUME
           INTO :HV_DOC_LOCATOR1
           FROM EMP_RESUME
          WHERE EMPNO = '000130'
            AND RESUME_FORMAT = 'ascii';
/*****************************************************/
/* Use the POSSTR function to locate the start of    */
/* sections "Department Information" and "Education" */   3
/*****************************************************/
EXEC SQL SET :HV_START_DEPTINFO =
          POSSTR(:HV_DOC_LOCATOR1, 'Department Information');

EXEC SQL SET :HV_START_EDUC =
             POSSTR(:HV_DOC_LOCATOR1, 'Education');
```

```
/*******************************************************/
/* Replace Department Information section with nothing */
/*******************************************************/
EXEC SQL SET :HV_DOC_LOCATOR2 =
      SUBSTR(:HV_DOC_LOCATOR1, 1, :HV_START_DEPTINFO -1)
   || SUBSTR (:HV_DOC_LOCATOR1, :HV_START_EDUC);
/*******************************************************/
```

```
/* Associate a new locator with the Department         */
/* Information section                                 */
/******************************************************/
EXEC SQL SET :HV_NEW_SECTION_LOCATOR =
    SUBSTR(:HV_DOC_LOCATOR1, :HV_START_DEPTINFO,
   :HV_START_EDUC -:HV_START_DEPTINFO);

/******************************************************/
/* Append the Department Information to the end       */
/* of the resume                                       */
/******************************************************/
EXEC SQL SET :HV_DOC_LOCATOR3 =
  :HV_DOC_LOCATOR2 || :HV_NEW_SECTION_LOCATOR;
/******************************************************/
/* Store the modified resume in the table.  This is    */  4
/* where the LOB data really moves.                    */
/******************************************************/
EXEC SQL INSERT INTO EMP_RESUME VALUES ('A00130', 'ascii',
   :HV_DOC_LOCATOR3, DEFAULT);

/*********************/
/* Free the locators */                                    5
/*********************/
EXEC SQL FREE LOCATOR :HV_DOC_LOCATOR1, :HV_DOC_LOCATOR2, :HV_DOC_LOCATOR3;
```

Notes:

1
Declare the LOB locators here.

2
This SELECT statement associates LOB locator HV_DOC_LOCATOR1 with the value of column RESUME for employee number 000130.

3
The next five SQL statements use LOB locators to manipulate the resume data without moving the data.

4
Evaluation of the LOB expressions in the previous statements has been deferred until execution of this INSERT statement.

5
Free all LOB locators to release them from their associated values.

LOB file reference variables

In a host application, you can use a file reference variable to insert a LOB or XML value from a file into a Db2 table. You can also use a file reference variable to select a LOB or XML value from a Db2 table into a file.

The file reference variables are BLOB_FILE, CLOB_FILE, or DBCLOB_FILE. For COBOL, the file reference variables are BLOB-FILE, CLOB-FILE, or DBCLOB-FILE.

When you use a file reference variable, you can select or insert an entire LOB or XML value without contiguous application storage to contain the entire LOB or XML value. LOB file reference variables move LOB or XML values from the database server to an application or from an application to the database server without going through the application's memory. Furthermore, LOB file reference variables bypass the host language limitation on the maximum size allowed for dynamic storage to contain a LOB value.

You can declare LOB or XML values as LOB file reference variables or LOB file reference arrays for applications that are written in C, COBOL, PL/I, and assembler. The LOB file reference variables do not contain LOB data; they represent a file that contains LOB data. Database queries, updates, and inserts can use file reference variables to store or retrieve column values. As with other host variables, a LOB file reference variable can have an associated indicator variable.

Db2-generated LOB file reference variable constructs

For each LOB file reference variable that an application declares for a LOB or XML value, Db2 generates an equivalent construct that uses the host language data types. When an application references a LOB file reference variable, it must use the equivalent construct that Db2 generates; otherwise the Db2 precompiler issues an error.

The construct describes the following properties of the file:

Data type
BLOB, CLOB, or DBCLOB. This property is specified when the variable is declared by using the BLOB_FILE, CLOB_FILE, or DBCLOB_FILE data type.

For COBOL, the data types are BLOB-FILE, CLOB-FILE, or DBCLOB-FILE.

Direction
This property must be specified by the application program at run time as part of the file option property. The direction property can have the following values:

Input
Used as a data source on an EXECUTE, OPEN, UPDATE, INSERT, DELETE, SET, or MERGE statement.

Output
Used as the target of data on a FETCH statement or a SELECT INTO statement.

File name
This property must be specified by the application program at run time. The file name property can have the following values:

- The complete path name of the file. This is recommended.

File name length
This property must be specified by the application program at run time.

File options
An application must assign one of the file options to a file reference variable before the application can use that variable. File options are set by the INTEGER value in a field in the file reference variable construct. One of the following values must be specified for each file reference variable:

- Input (from application to database):

  SQL_FILE_READ
  A regular file that can be opened, read, and closed.

- Output (from database to application):

  SQL_FILE_CREATE
  If the file does not exists, a new file is created. If the file already exists, an error is returned.

  SQL_FILE_OVERWRITE
  If the file does not exists, a new file is created. If the file already exists, it is overwritten.

  SQL_FILE_APPEND
  If the file does not exists, a new file is created. If the file already exists, the output is appended to the existing file.

  Data length
  The length, in bytes, of the new data written to the file

Examples of declaring file reference variables

You can declare a file reference variable in C, COBOL, and PL/I, and declare the file reference variable construct that Db2 generates.

C Example: Consider the following C declaration:

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS CLOB_FILE hv_text_file;
  CHAR hv_thesis_title[64];
EXEC SQL END DECLARE SECTION
```

That declaration results in the following Db2-generated construct:

```
EXEC SQL BEGIN DECLARE SECTION
    /* SQL TYPE IS CLOB_FILE hv_text_file; */
  struct {
     unsigned long name_length;  // File name length
     unsigned long data_length;  // Data length
     unsigned long file_options; // File options
     char name [255];            // File name
         } hv_text_file;
  char hv_thesis_title[64]
```

With the Db2-generated construct, you can use the following code to select from a CLOB column in the database into a new file that is referenced by `:hv_text_file`. The file name must be an absolute path.

```
strcopy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
        hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
        hv_text_file.file_options = SQL_FILE_CREATE;
```

```
EXEC SQL SELECT CONTENT INTO :hv_text_file FROM PAPERS
           WHERE TITLE = 'The Relational Theory Behind Juggling';
```

Similarly, you can use the following code to insert the data from a file that is referenced by `:hv_text_file` into a CLOB column. The file name must be an absolute path.

```
strcopy(hv_text_file.name, "/u/gainer/patents/chips.13");
        hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
        hv_text_file.file_options = SQL_FILE_READ;
strcopy(:hv_patent_title, "A Method for Pipelining Chip Consumption");
```

```
EXEC SQL INSERT INTO PATENTS(TITLE, TEXT)
                VALUES(:hv_patent_title, :hv_text_file);
```

COBOL Example: Consider the following COBOL declaration:

```
01 MY-FILE SQL TYPE IS BLOB-FILE
```

That declaration results in the following Db2-generated construct:

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTION PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC(255);
```

PL/I Example: Consider the following PL/I declaration:

```
DCL MY_FILE SQL TYPE IS CLOB_FILE
```

That declaration results in the following Db2-generated construct:

```
DCL 1 MY_FILE,
   3 MY_FILE_NAME_LENGTH BINARY FIXED (31) UNALIGNED,
   3 MY_FILE_DATA_LENGTH BINARY FIXED (31) UNALIGNED,
   3 MY_FILE_FILE_OPTIONS BINARY FIXED (31) UNALIGNED,
   3 MY_FILE_NAME CHAR(255);
```

For examples of how to declare file reference variables for XML data in C, COBOL, and PL/I, see “Host variable data types for XML data in embedded SQL applications” on page 542.

Referencing a sequence object

A *sequence* object is a user-defined object that generates a sequence of numeric values according to the specification with which the sequence was created. You can retrieve the next or previous value in the sequence.

About this task

You reference a sequence by using the NEXT VALUE expression or the PREVIOUS VALUE expression, specifying the name of the sequence:

- A NEXT VALUE expression generates and returns the next value for the specified sequence. If a query contains multiple instances of a NEXT VALUE expression with the same sequence name, the sequence value increments only once for that query. The ROLLBACK statement has no effect on values already generated.
- A PREVIOUS VALUE expression returns the most recently generated value for the specified sequence for a previous NEXT VALUE expression that specified the same sequence within the current application process. The value of the PREVIOUS VALUE expression persists until the next value is generated for the sequence, the sequence is dropped, or the application session ends. The COMMIT statement and the ROLLBACK statement have no effect on this value.

You can specify a NEXT VALUE or PREVIOUS VALUE expression in a SELECT clause, within a VALUES clause of an insert operation, within the SET clause of an update operation (with certain restrictions), or within a SET *host-variable* statement.

Retrieving thousands of rows

When retrieving large numbers of rows, consider the possibilities for lock escalation and other locking issues.

About this task

Question: Are there any special techniques for fetching and displaying large volumes of data?

Answer: There are no special techniques; but for large numbers of rows, efficiency can become very important. In particular, you need to be aware of locking considerations, including the possibilities of lock escalation.

If your program allows input from a terminal before it commits the data and thereby releases locks, it is possible that a significant loss of concurrency results.

Determining when a row was changed

If a table has a ROW CHANGE TIMESTAMP column, you can determine when a row was changed.

Procedure

Issue a SELECT statement with the ROW CHANGE TIMESTAMP column in the column list.

If a qualifying row does not have a value for the ROW CHANGE TIMESTAMP column, Db2 returns the time that the page in which that row resides was updated.

Example

Suppose that you issue the following statements to create, populate, and alter a table:

```
CREATE TABLE T1 (C1 INTEGER NOT NULL);
INSERT INTO T1 VALUES (1);
ALTER TABLE T1 ADD COLUMN C2 NOT NULL GENERATED ALWAYS
  FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP;
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Because the ROW CHANGE TIMESTAMP column was added after the data was inserted, the following statement returns the time that the page was last modified:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Assume that you then issue the following statement:

```
INSERT INTO T1(C1) VALUES (2);
```

Assume that this row is added to the same page as the first row. The following statement returns the time that value "2" was inserted into the table:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 2;
```

Because the row with value "1" still does not have a value for the ROW CHANGE TIMESTAMP column, the following statement still returns the time that the page was last modified, which in this case is the time that value "2" was inserted:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Related reference

CREATE TABLE (Db2 SQL)

Checking whether an XML column contains a certain value

You can determine which rows contain any fragment of XML data that you specify.

Procedure

Specify the XMLEXISTS predicate in the WHERE clause of your SQL statement.

Include the following parameters for the XMLEXISTS predicate:

- An XPath expression that is embedded in a character string literal. Specify an XPath expression that identifies the XML data that you are looking for. If the result of the XPath expression is an empty sequence, XMLEXISTS returns false. If the result is not empty, XMLEXISTS returns true. If the evaluation of the XPath expression returns an error, XMLEXISTS returns an error.
- The XML column name. Specify this value after the PASSING keyword.

Example

Suppose that you want to return only purchase orders that have a billing address. Assume that column XMLPO stores the XML purchase order documents and that the `billTo` nodes within these documents contain any billing addresses. You can use the following SELECT statement with the XMLEXISTS predicate:

```
SELECT XMLPO FROM T1
  WHERE XMLEXISTS ('declare namespace ipo="http://www.example.com/IPO";
                    /ipo:purchaseOrder[billTo]'
                    PASSING XMLPO);
```

Related reference

XMLEXISTS predicate (Db2 SQL)

Accessing Db2 data that is not in a table

You can access Db2 data that is not in a table by returning the value of an SQL expression in a host variable.

Procedure

To return the value of an SQL expression that does not include the value of a table column in host variable, use one of the following approaches:

- Use the SET host-variable assignment statement to set the contents of a host variable to the value of an expression.

  ```
  EXEC SQL SET :hvrandval = RAND(:hvrand);
  ```

- Use the VALUES INTO statement to return the value of an expression in a host variable.

  ```
  EXEC SQL VALUES RAND(:hvrand)
     INTO :hvrandval;
  ```

- Use the following statement to select the expression from the Db2-provided EBCDIC table, named SYSIBM.SYSDUMMY1, which consists of one row.

  ```
  EXEC SQL SELECT RAND(:hvrand)
     INTO :hvrandval
     FROM SYSIBM.SYSDUMMY1;
  ```

Related reference

SET assignment-statement (Db2 SQL)
VALUES INTO (Db2 SQL)
SYSDUMMY1 catalog table (Db2 SQL)

Ensuring that queries perform sufficiently

It is important to make sure that any individual queries that are included in your program are not slowing down the performance of your program.

About this task

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- IBM Data Server Manager (Introduction to Db2 for z/OS)
- Db2 Tools for z/OS

Procedure

To ensure that queries perform sufficiently:

1. Tune each query in your program by following the general tuning guidelines for how to write efficient queries.
2. If you suspect that a query is not as efficient as it could be, monitor its performance.

   You can use a number of different functions and techniques to monitor SQL performance, including the SQL EXPLAIN statement and SQL optimization tools.

Related concepts

Investigating SQL performance by using EXPLAIN (Db2 Performance)
Interpreting data access by using EXPLAIN (Db2 Performance)

Related tasks

Programming applications for performance (Db2 Performance)
Investigating access path problems (Db2 Performance)

Related reference
EXPLAIN (Db2 SQL)
Related information
Tuning SQL with Optim Query Tuner, Part 1: Understanding access paths (IBM Data Studio)

Items to include in a batch DL/I program

When you use a batch DL/I program with Db2, you must include certain items in your program.

A batch DL/I program can issue:

- Any IMS batch call, except ROLS, SETS, and SYNC calls. ROLS and SETS calls provide intermediate backout point processing, which Db2 does not support. The SYNC call provides commit point processing without identifying the commit point with a value. IMS does not allow a SYNC call in batch, and neither does the Db2 DL/I batch support.

  Issuing a ROLS, SETS, or SYNC call in an application program causes a system abend X'04E' with the reason code X'00D44057' in register 15.
- GSAM calls.
- IMS system services calls.
- Any SQL statements, except COMMIT and ROLLBACK. IMS and CICS environments do not allow those SQL statements; however, IMS and CICS do allow ROLLBACK TO SAVEPOINT. You can use the IMS CHKP call to commit data and the IMS ROLL or ROLB to roll back changes.

  Issuing a COMMIT statement causes SQLCODE -925; issuing a ROLLBACK statement causes SQLCODE -926. Those statements also return SQLSTATE '2D521'.
- Any call to a standard or traditional access method (for example, QSAM, VSAM, and so on).

The restart capabilities for Db2 and IMS databases, as well as for sequential data sets that are accessed through GSAM, are available through the IMS Checkpoint and Restart facility.

Db2 allows access to both Db2 and DL/I data through the use of the following Db2 and IMS facilities:

- IMS synchronization calls, which commit and abnormally terminate units of recovery
- The Db2 IMS attachment facility, which handles the two-phase commit protocol and enables both systems to synchronize a unit of recovery during a restart after a failure
- The IMS log, which is used to record the instant of commit

In a data sharing environment, DL/I batch supports group attachment or subgroup attachment. You can specify a group attachment name instead of a subsystem name in the SSN parameter of the DDITV02 data set for the DL/I batch job.

Requirements for using Db2 in a DL/I batch job

Using Db2 in a DL/I batch job requires the following changes to the application program and the job step JCL:

- Add SQL statements to your application program to gain access to Db2 data. You must then precompile the application program and bind the resulting DBRM into a package.
- Before you run the application program, use JOBLIB, STEPLIB, or link book to access the Db2 load library, so that Db2 modules can be loaded.
- In a data set that is specified by a DDITV02 DD statement, specify the program name and plan name for the application, and the connection name for the DL/I batch job.

  In an input data set or in a subsystem member, specify information about the connection between Db2 and IMS. The input data set name is specified with a DDITV02 DD statement. The subsystem member name is specified by the parameter SSM= on the DL/I batch invocation procedure.
- Optionally specify an output data set using the DDOTV02 DD statement. You might need this data set to receive messages from the IMS attachment facility about indoubt threads and diagnostic information.

Program design considerations for using DL/I batch

Address spaces in DL/I batch:
A DL/I batch region is independent of both the IMS control region and the CICS address space. The DL/I batch region loads the DL/I code into the application region along with the application program.

Commits in DL/I batch:
Commit IMS batch applications frequently so that you do not use resources for an extended time.

SQL statements and IMS calls in DL/I batch:
DL/I batch applications cannot use the SQL COMMIT and ROLLBACK statements; otherwise, you get an SQL error code. DLI/I batch applications also cannot use ROLS, SETS, and SYNC calls; otherwise the application program abnormally terminates.

Checkpoint calls in DL/I batch:
Write your program with SQL statements and DL/I calls, and use checkpoint calls. The frequency of checkpoints depends on the application design. All checkpoints that are issued by a batch application program must be unique. At a checkpoint, DL/I positioning is lost, Db2 cursors are closed (with the possible exception of cursors that are defined as WITH HOLD), commit duration locks are freed (again with some exceptions), and database changes are considered permanent to both IMS and Db2.

Application program synchronization in DL/I batch:
You can design an application program without using IMS checkpoints. In that case, if the program abnormally terminates before completing, Db2 backs out any updates, and you can use the IMS batch backout utility to back out the DL/I changes.

You can also have IMS dynamically back out the updates within the same job. You must specify the BKO parameter as 'Y' and allocate the IMS log to DASD.

You could have a problem if the system on which the job is run fails after the program terminates but before the job step ends. If you do not have a checkpoint call before the program ends, Db2 commits the unit of work without involving IMS. If the system fails before DL/I commits the data, the Db2 data is out of synchronization with the DL/I changes. If the system fails during Db2 commit processing, the Db2 data could be indoubt. When you restart the application program, use the XRST call to obtain checkpoint information and resolve any Db2 indoubt work units.

Recommendation: Always issue a symbolic checkpoint at the end of any update job to coordinate the commit of the outstanding unit of work for IMS and Db2.

Checkpoint and XRST considerations in DL/I batch:
If you use an XRST call, Db2 assumes that any checkpoint that is issued is a symbolic checkpoint. The options of the symbolic checkpoint call differ from the options of a basic checkpoint call. Using the incorrect form of the checkpoint call can cause problems.

If you do not use an XRST call, Db2 assumes that any checkpoint call that is issued is a basic checkpoint.

To make restart easier, use EBCDIC characters for checkpoint IDs.

When an application program needs to be restartable, you must use symbolic checkpoint and XRST calls. If you use an XRST call, it must be the first IMS call that is issued, and it must occur before any SQL statement. Also, you must use only one XRST call.

Synchronization call abends in DL/I batch:
If the application program contains an incorrect IMS synchronization call (CHKP, ROLB, ROLL, or XRST), causing IMS to issue a bad status code in the PCB, Db2 abends the application program. Be sure to test these calls before placing the programs in production.

Related concepts

Input and output data sets for DL/I batch jobs
DL/I batch jobs require an input data set with DD name DDITV02 and an output data set with DD name DDOTV02.

Multiple system consistency (Db2 Administration Guide)

Related tasks

Preparing an application to run on Db2 for z/OS

To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

Invoking a user-defined function

You can use a user-defined function wherever you can use a built-in function.

You can invoke a sourced or external user-defined scalar function in an SQL statement wherever you use an expression. For a table function, you can invoke the user-defined function only in the FROM clause of a SELECT statement. The invoking SQL statement can be in a stand alone program, a stored procedure, a trigger body, or another user-defined function.

Recommendations for invoking user-defined functions:

Invoke user-defined functions with external actions and nondeterministic user-defined functions from select lists: Invoking user-defined functions with external action from a select list and nondeterministic user-defined functions from a select list is preferred to invoking these user-defined functions from a predicate.

The access path that Db2 chooses for a predicate determines whether a user-defined function in that predicate is invoked. To ensure that Db2 executes the external action for each row of the result table, put the user-defined function invocation in the SELECT list.

Invoking a nondeterministic user-defined function from a predicate can yield undesirable results. The following example demonstrates this idea.

Suppose that you execute this query:

```
SELECT COUNTER(), C1, C2 FROM T1 WHERE COUNTER() = 2;
```

Table T1 looks like this:

```
C1  C2
--  --
 1   b
 2   c
 3   a
```

COUNTER is a user-defined function that increments a variable in the scratchpad each time it is invoked.

Db2 invokes an instance of COUNTER in the predicate 3 times. Assume that COUNTER is invoked for row 1 first, for row 2 second, and for row 3 third. Then COUNTER returns 1 for row 1, 2 for row 2, and 3 for row 3. Therefore, row 2 satisfies the predicate WHERE COUNTER()=2, so Db2 evaluates the SELECT list for row 2. Db2 uses a different instance of COUNTER in the select list from the instance in the predicate. Because the instance of COUNTER in the select list is invoked only once, it returns a value of 1. Therefore, the result of the query is:

```
COUNTER()  C1 C2
---------  -- --
    1       2  c
```

This is not the result you might expect.

The results can differ even more, depending on the order in which Db2 retrieves the rows from the table. Suppose that an ascending index is defined on column C2. Then Db2 retrieves row 3 first, row 1 second, and row 2 third. This means that row 1 satisfies the predicate WHERE COUNTER()=2. The value of COUNTER in the select list is again 1, so the result of the query in this case is:

```
COUNTER()  C1 C2
---------  -- --
    1       1  b
```

Understand the interaction between scrollable cursors and nondeterministic user-defined functions or user-defined functions with external actions: When you use a scrollable cursor, you might retrieve the same row multiple times while the cursor is open. If the select list of the cursor's SELECT statement contains a user-defined function, that user-defined function is executed each time you retrieve a row. Therefore, if the user-defined function has an external action, and you retrieve the same row multiple times, the external action is executed multiple times for that row.

A similar situation occurs with scrollable cursors and nondeterministic functions. The result of a nondeterministic user-defined function can be different each time you execute the user-defined function. If the select list of a scrollable cursor contains a nondeterministic user-defined function, and you use that cursor to retrieve the same row multiple times, the results can differ each time you retrieve the row.

A nondeterministic user-defined function in the predicate of a scrollable cursor's SELECT statement does not change the result of the predicate while the cursor is open. Db2 evaluates a user-defined function in the predicate only once while the cursor is open.

Related concepts

Abnormal termination of an external user-defined function
If an external user-defined function abnormally terminates, your program receives SQLCODE -430 for invoking the statement.

Function invocation (Db2 SQL)

Related reference

from-clause (Db2 SQL)

How Db2 determines the authorization for invoking user-defined functions

Both the authorization used to invoke a user-defined function and the authorization used for executing each SQL statement in the function influence the processing of a user-defined function.

The authorization that is required to invoke a user defined function depends on the whether the function is invoked statically or dynamically:

- For static invocations, the authorization of the owner of the package that contains the invocation of the user-defined function is used.
- For dynamic invocations, the DYNAMICRULES bind option of the package that contains the invocation of the user-defined determines the authorization that is used. For more information about how Db2 applies the DYNAMICRULES bind option, see DYNAMICRULES bind option (Db2 Commands).

Similarly, the authorization that Db2 uses to process each SQL statement inside a user-defined function depends on whether the statement is a static or dynamic SQL statement:

- For static SQL statements, the authorization of the owner of the user-defined function is used.
- For dynamic SQL statements, the DYNAMICRULES option of the CREATE FUNCTION statement determines the authorization that is used.

Related concepts

Function invocation (Db2 SQL)

Related reference

CREATE FUNCTION (Db2 SQL)

Ensuring that Db2 executes the intended user-defined function

Multiple functions can with the same name can exist in the same schema or in different schemas. You can take certain actions so that Db2 chooses the correct function to execute.

About this task

The combination of the function name and the parameter list form the *signature* that Db2 uses to identify a function. For detailed information about the rules and process that Db2 uses to identify the function to invoke, see Function resolution (Db2 SQL).

If the signatures of two functions match, including built-in and user-defined functions, you must take appropriate action to ensure that Db2 invokes the correct intended function.

Procedure

To simplify the resolution of built-in and user-defined functions, use the following techniques:

- When you invoke a function, use the qualified name.

  This causes Db2 to search for functions only in the schema you specify. This approach has the following advantages:

  - Db2 is less likely to choose a function that you did not intend to use. Several functions might fit the invocation equally well. Db2 picks the function whose schema name is listed first in the SQL path, which might not be the function you want.
  - The number of candidate functions is smaller, so Db2 takes less time for function resolution.
- Cast parameters in a user-defined function invocation to the types in the user-defined function definition. For example, if an input parameter for user-defined function FUNC is defined as DECIMAL(13,2), and the value you want to pass to the user-defined function is an integer value, cast the integer value to DECIMAL(13,2):

  For example, if an input parameter for user-defined function FUNC is defined as DECIMAL(13,2), and the value you want to pass to the user-defined function is an integer value, cast the integer value to DECIMAL(13,2):

  ```
  SELECT FUNC(CAST (INTCOL AS DECIMAL(13,2))) FROM T1;
  ```
- Use the data type BIGINT for numeric parameters in a user-defined function.

  When you invoke the function, you can pass in SMALLINT, INTEGER, or BIGINT values. If you use SMALLINT or REAL as the parameter type, you must pass parameters of the same types. For example, if user-defined function FUNC is defined with a parameter of type SMALLINT, only an invocation with a parameter of type SMALLINT resolves correctly. The following call does not resolve to FUNC because the constant 123 is of type INTEGER, not SMALLINT:

  ```
  SELECT FUNC(123) FROM T1;
  ```
- Avoid defining user-defined function string parameters with fixed-length string types.

  If you define a parameter with a fixed-length string type (CHAR, GRAPHIC, or BINARY), you can invoke the user-defined function only with a fixed-length string parameter. However, if you define the parameter with a varying-length string type (VARCHAR, VARGRAPHIC, or VARBINARY), you can invoke the user-defined function with either a fixed-length string parameter or a varying-length string parameter.

  If you must define parameters for a user-defined function as CHAR or BINARY, and you call the user-defined function from a C program or SQL procedure, you need to cast the corresponding parameter values in the user-defined function invocation to CHAR or BINARY to ensure that Db2 invokes the correct function. For example, suppose that a C program calls user-defined function CVRTNUM, which takes one input parameter of type CHAR(6). Also suppose that you declare host variable empnumbr as `char empnumbr[6]`. When you invoke CVRTNUM, cast empnumbr to CHAR:

  ```
  UPDATE EMP
   SET EMPNO=CVRTNUM(CHAR(:empnumbr))
   WHERE EMPNO = :empnumbr;
  ```

Related concepts

Functions (Db2 SQL)

How Db2 resolves functions

Function resolution is the process by which Db2 determines which user-defined function or built-in function to execute. You need to understand the function resolution process that Db2 uses to ensure that you invoke the user-defined function that you want to invoke.

Several user-defined functions with the same name but different numbers or types of parameters can exist in a Db2 subsystem. Several user-defined functions with the same name can have the same number of parameters, as long as the data types of any of the first 30 parameters are different. In addition, several user-defined functions might have the same name as a built-in function. When you invoke a function, Db2 must determine which user-defined function or built-in function to execute.

Db2 performs these steps for function resolution:

1. Determines if any function instances are candidates for execution. If no candidates exist, Db2 issues an SQL error message.
2. Compares the data types of the input parameters to determine which candidates fit the invocation best.

   Db2 does not compare data types for input parameters that are untyped parameter markers.

   For a qualified function invocation, if there are no parameter markers in the invocation, the result of the data type comparison is one best fit. That best fit is the choice for execution. If there are parameter markers in the invocation, there might be more than one best fit. Db2 issues an error if there is more than one best fit.

   For an unqualified function invocation, Db2 might find multiple best fits because the same function name with the same input parameters can exist in different schemas, or because there are parameter markers in the invocation.
3. If two or more candidates fit the unqualified function invocation equally well because the same function name with the same input parameters exists in different schemas, Db2 chooses the user-defined function whose schema name is earliest in the SQL path.

   For example, suppose functions SCHEMA1.X and SCHEMA2.X fit a function invocation equally well. Assume that the SQL path is:

   ```
   "SCHEMA2", "SYSPROC", "SYSIBM", "SCHEMA1", "SYSFUN"
   ```

   Then Db2 chooses function SCHEMA2.X.

   If two or more candidates fit the unqualified function invocation equally well because the function invocation contains parameter markers, Db2 issues an error.

The remainder of this section discusses details of the function resolution process and gives suggestions on how you can ensure that Db2 picks the right function.

How Db2 chooses candidate functions:

An instance of a user-defined function is a candidate for execution only if it meets all of the following criteria:

- If the function name is qualified in the invocation, the schema of the function instance matches the schema in the function invocation.

  If the function name is unqualified in the invocation, the schema of the function instance matches a schema in the invoker's SQL path.
- The name of the function instance matches the name in the function invocation.
- The number of input parameters in the function instance matches the number of input parameters in the function invocation.
- The function invoker is authorized to execute the function instance.
- The type of each of the input parameters in the function invocation matches or is *promotable* to the type of the corresponding parameter in the function instance.

If an input parameter in the function invocation is an untyped parameter marker, Db2 considers that parameter to be a match or promotable.

For a function invocation that passes a transition table, the data type, length, precision, and scale of each column in the transition table must match exactly the data type, length, precision, and scale of each column of the table that is named in the function instance definition. For information about transition tables, see "Creating a trigger" on page 151.

- The create timestamp for a user-defined function must be older than the BIND or REBIND timestamp for the package or plan in which the user-defined function is invoked.

  If Db2 authorization checking is in effect, and Db2 performs an automatic rebind on a plan or package that contains a user-defined function invocation, any user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

  If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package.

  If a user-defined function is invoked during an automatic rebind, and that user-defined function is invoked from a trigger body and receives a transition table, then the form of the invoked function that Db2 uses for function selection includes only the columns of the transition table that existed at the time of the original BIND or REBIND of the package or plan for the invoking program.

  During an automatic rebind, Db2 does not consider built-in functions for function resolution if those built-in functions were introduced in a later release of Db2 than the release in which the BIND or REBIND of the invoking plan or package occurred.

  When you explicitly bind or rebind a plan or package, the plan or package receives a release dependency marker. When Db2 performs an automatic rebind of a query that contains a function invocation, a built-in function is a candidate for function resolution only if the release dependency marker of the built-in function is the same as or lower than the release dependency marker of the plan or package that contains the function invocation.

Example: Suppose that in this statement, the data type of A is SMALLINT:

```
SELECT USER1.ADDTWO(A) FROM TABLEA;
```

Two instances of USER1.ADDTWO are defined: one with an input parameter of type INTEGER and one with an input parameter of type DECIMAL. Both function instances are candidates for execution because the SMALLINT type is promotable to either INTEGER or DECIMAL. However, the instance with the INTEGER type is a better fit because INTEGER is higher in the list than DECIMAL.

How Db2 chooses the best fit among candidate functions:

More than one function instance might be a candidate for execution. In that case, Db2 determines which function instances are the best fit for the invocation by comparing parameter data types.

If the data types of all parameters in a function instance are the same as those in the function invocation, that function instance is a best fit. If no exact match exists, Db2 compares data types in the parameter lists from left to right, using this method:

1. Db2 compares the data types of the first parameter in the function invocation to the data type of the first parameter in each function instance.

   If the first parameter in the invocation is an untyped parameter marker, Db2 does not do the comparison.
2. For the first parameter, if one function instance has a data type that fits the function invocation better than the data types in the other instances, that function is a best fit.
3. If the data types of the first parameter are the same for all function instances, or if the first parameter in the function invocation is an untyped parameter marker, Db2 repeats this process for the next parameter. Db2 continues this process for each parameter until it finds a best fit.

Example of function resolution: Suppose that a program contains the following statement:

```
SELECT FUNC(VCHARCOL,SMINTCOL,DECCOL) FROM T1;
```

In user-defined function FUNC, VCHARCOL has data type VARCHAR, SMINTCOL has data type SMALLINT, and DECCOL has data type DECIMAL. Also suppose that two function instances with the following definitions meet the appropriate criteria and are therefore candidates for execution.

```
Candidate 1:
CREATE FUNCTION FUNC(VARCHAR(20),INTEGER,DOUBLE)
  RETURNS DECIMAL(9,2)
  EXTERNAL NAME 'FUNC1'
  PARAMETER STYLE SQL
  LANGUAGE COBOL;

Candidate 2:
CREATE FUNCTION FUNC(VARCHAR(20),REAL,DOUBLE)
  RETURNS DECIMAL(9,2)
  EXTERNAL NAME 'FUNC2'
  PARAMETER STYLE SQL
  LANGUAGE COBOL;
```

Db2 compares the data type of the first parameter in the user-defined function invocation to the data types of the first parameters in the candidate functions. Because the first parameter in the invocation has data type VARCHAR, and both candidate functions also have data type VARCHAR, Db2 cannot determine the better candidate based on the first parameter. Therefore, Db2 compares the data types of the second parameters.

The data type of the second parameter in the invocation is SMALLINT. INTEGER, which is the data type of candidate 1, is a better fit to SMALLINT than REAL, which is the data type of candidate 2. Therefore, candidate 1 is the Db2 choice for execution.

Related concepts

Promotion of data types (Db2 SQL)

Related tasks

Creating a trigger

A *trigger* is a set of SQL statements that execute when a certain event occurs in a table or view. Use triggers to control changes in Db2 databases. Triggers are more powerful than constraints because they can monitor a broader range of changes and perform a broader range of actions.

Related information

Exit routines (Db2 Administration Guide)

Checking how Db2 resolves functions by using DSN_FUNCTION_TABLE

Because multiple user-defined functions can have the same name, you should ensure that Db2 invokes the function that you intended to invoke. One way to check that the correct function was invoked is to use a function table called DSN_FUNCTION_TABLE.

Procedure

To check how Db2 resolves a function by using DSN_FUNCTION_TABLE:

1. If *your_userID*.DSN_FUNCTION_TABLE does not already exist, create this table by following the instructions in DSN_FUNCTION_TABLE (Db2 Performance).
2. Populate *your_userID*.DSN_FUNCTION_TABLE with information about which functions are invoked by a particular SQL statement by performing one of the following actions:

   - Execute the EXPLAIN statement on the SQL statement.
   - Ensure that the program that contains the SQL statement is bound with EXPLAIN(YES) and run the program.

   Db2 puts a row in *your_userID*.DSN_FUNCTION_TABLE for each function that is referenced in each SQL statement.

3. Check the rows that were added to *your_userID*.DSN_FUNCTION_TABLE to ensure that the appropriate function was invoked. Use the following columns to help you find applicable rows: QUERYNO, APPLNAME, PROGNAM, COLLID, and EXPLAIN_TIME.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

EXPLAIN (Db2 SQL)

Restrictions when passing arguments with distinct types to functions

Because Db2 enforces strong typing when you pass arguments to a function, you must follow certain rules when passing arguments with distinct types to functions.

Adhere to the following rules:

- You can pass arguments that have distinct types to a function if either of the following conditions is true:
  - A version of the function that accepts those distinct types is defined.

    This also applies to infix operators. If you want to use one of the five built-in infix operators (||, /, *, +, -) with your distinct types, you must define a version of that operator that accepts the distinct types.
  - You can cast your distinct types to the argument types of the function.
- If you pass arguments to a function that accepts only distinct types, the arguments you pass must have the same distinct types as in the function definition. If the types are different, you must cast your arguments to the distinct types in the function definition.

  If you pass constants or host variables to a function that accepts only distinct types, you must cast the constants or host variables to the distinct types that the function accepts.

The following examples demonstrate how to use distinct types as arguments in function invocations.

Example: Defining a function with distinct types as arguments: Suppose that you want to invoke the built-in function HOUR with a distinct type that is defined like this:

```
CREATE DISTINCT TYPE FLIGHT_TIME AS TIME;
```

The HOUR function takes only the TIME or TIMESTAMP data type as an argument, so you need a sourced function that is based on the HOUR function that accepts the FLIGHT_TIME data type. You might declare a function like this:

```
CREATE FUNCTION HOUR(FLIGHT_TIME)
  RETURNS INTEGER
  SOURCE SYSIBM.HOUR(TIME);
```

Example: Casting function arguments to acceptable types: Another way you can invoke the HOUR function is to cast the argument of type FLIGHT_TIME to the TIME data type before you invoke the HOUR function. Suppose table FLIGHT_INFO contains column DEPARTURE_TIME, which has data type FLIGHT_TIME, and you want to use the HOUR function to extract the hour of departure from the departure time. You can cast DEPARTURE_TIME to the TIME data type, and then invoke the HOUR function:

```
SELECT HOUR(CAST(DEPARTURE_TIME AS TIME)) FROM FLIGHT_INFO;
```

Example: Using an infix operator with distinct type arguments: Suppose you want to add two values of type US_DOLLAR. Before you can do this, you must define a version of the + function that accepts values of type US_DOLLAR as operands:

```
CREATE FUNCTION "+"(US_DOLLAR,US_DOLLAR)
  RETURNS US_DOLLAR
  SOURCE SYSIBM."+"(DECIMAL(9,2),DECIMAL(9,2));
```

Because the US_DOLLAR type is based on the DECIMAL(9,2) type, the source function must be the version of + with arguments of type DECIMAL(9,2).

Example: Casting constants and host variables to distinct types to invoke a user-defined function: Suppose function CDN_TO_US is defined like this:

```
CREATE FUNCTION EURO_TO_US(EURO)
  RETURNS US_DOLLAR
  EXTERNAL NAME 'CDNCVT'
  PARAMETER STYLE SQL
  LANGUAGE C;
```

This means that EURO_TO_US accepts only the EURO type as input. Therefore, if you want to call CDN_TO_US with a constant or host variable argument, you must cast that argument to distinct type EURO:

```
SELECT * FROM US_SALES
  WHERE TOTAL = EURO_TO_US(EURO(:H1));
```

```
SELECT * FROM US_SALES
  WHERE TOTAL = EURO_TO_US(EURO(10000));
```

Cases when Db2 casts arguments for a user-defined function

In certain situations, when you invoke a user-defined function, Db2 casts your input argument values to different data types and lengths.

Whenever you invoke a user-defined function, Db2 assigns your input argument values to parameters with the data types and lengths in the user-defined function definition.

When you invoke a user-defined function that is sourced on another function, Db2 casts your arguments to the data types and lengths of the sourced function.

The following example demonstrates what happens when the parameter definitions of a sourced function differ from those of the function on which it is sourced.

Suppose that external user-defined function TAXFN1 is defined like this:

```
CREATE FUNCTION TAXFN1(DEC(6,0))
  RETURNS DEC(5,2)
  PARAMETER STYLE SQL
  LANGUAGE C
  EXTERNAL NAME TAXPROG;
```

Sourced user-defined function TAXFN2, which is sourced on TAXFN1, is defined like this:

```
CREATE FUNCTION TAXFN2(DEC(8,2))
  RETURNS DEC(5,0)
  SOURCE TAXFN1;
```

You invoke TAXFN2 using this SQL statement:

```
UPDATE TB1
  SET SALESTAX2 = TAXFN2(PRICE2);
```

TB1 is defined like this:

```
CREATE TABLE TB1
  (PRICE1    DEC(6,0),
   SALESTAX1 DEC(5,2),
   PRICE2    DEC(9,2),
   SALESTAX2 DEC(7,2));
```

Now suppose that PRICE2 has the DECIMAL(9,2) value 0001234.56. Db2 must first assign this value to the data type of the input parameter in the definition of TAXFN2, which is DECIMAL(8,2). The input parameter value then becomes 001234.56. Next, Db2 casts the parameter value to a source function parameter, which is DECIMAL(6,0). The parameter value then becomes 001234. (When you cast a value, that value is truncated, rather than rounded.)

Now, if TAXFN1 returns the DECIMAL(5,2) value 123.45, Db2 casts the value to DECIMAL(5,0), which is the result type for TAXFN2, and the value becomes 00123. This is the value that Db2 assigns to column SALESTAX2 in the UPDATE statement.

Casting of parameter markers

You can use untyped parameter markers in a function invocation. However, Db2 cannot compare the data types of untyped parameter markers to the data types of candidate functions. Therefore, Db2 might find more than one function that qualifies for invocation. If this happens, an SQL error occurs. To ensure that Db2 picks the right function to execute, cast the parameter markers in your function invocation to the data types of the parameters in the function that you want to execute. For example, suppose that two versions of function FX exist. One version of FX is defined with a parameter of type of DECIMAL(9,2), and the other is defined with a parameter of type INTEGER. You want to invoke FX with a parameter marker, and you want Db2 to execute the version of FX that has a DECIMAL(9,2) parameter. You need to cast the parameter marker to a DECIMAL(9,2) type by using a CAST specification:

```
SELECT FX(CAST(? AS DECIMAL(9,2))) FROM T1;
```

Related concepts

Assignment and comparison (Db2 SQL)

Chapter 4. Embedded SQL programming

Application programs written in host languages such as COBOL can contain SQL statements. The source form of a static SQL statement is embedded within application program, and the statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

An application program can also contain dynamic SQL statements referred to as embedded dynamic SQL. Programs that contain embedded dynamic SQL statements must be precompiled like those that contain static SQL, but unlike static SQL, the dynamic statements are constructed and prepared at run time. The source form of a dynamic statement is a character string that is passed to Db2 by the program using the static SQL PREPARE or EXECUTE IMMEDIATE statement. A statement that is prepared using the PREPARE statement can be referenced in a DECLARE CURSOR, DESCRIBE, or EXECUTE statement. Whether the operational form of the statement is persistent depends on whether dynamic statement caching is enabled.

Overview of programming applications that access Db2 for z/OS data

Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

About this task

A *query* is an SQL statement that returns data from a Db2 database. Your program can use several methods to communicate SQL statements to Db2 for z/OS. After processing the statement, Db2 issues a return code, which your program can test to determine the result of the operation.

Introductory concepts

Programming for Db2 for z/OS (Introduction to Db2 for z/OS)
Tools and IDEs for developing Db2 applications (Introduction to Db2 for z/OS)
Preparation process for an application program (Introduction to Db2 for z/OS)
Performance information for SQL application programming (Introduction to Db2 for z/OS)

Procedure

To include Db2 for z/OS queries in an application program:

1. Choose one of the following methods for communicating with Db2:

   Static SQL
   The source form of a static SQL statement is embedded within an application program written in a host language. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

   Embedded dynamic SQL
   Dynamic SQL is prepared and executed while the program is running.

   Open Database Connectivity (ODBC)
   You access data through ODBC function calls in your application. You execute SQL statements by passing them to Db2 through a ODBC function call. ODBC eliminates the need for precompiling and binding your application and increases the portability of your application by using the ODBC interface.

   JDBC application support
   If you are writing your applications in Java, you can use JDBC application support to access Db2. JDBC is similar to ODBC but is designed specifically for use with Java.

SQLJ application support
: You also can use SQLJ application support to access Db2. SQLJ is designed to simplify the coding of Db2 calls for Java applications.

Db2 for Linux, UNIX, and Windows drivers
: You can use the client drivers to connect to Db2 for z/OS from application programming languages such as Node.js, Perl, Python, Ruby on Rails, PHP, and others.

2. Optional: Declare the tables and views that you use.

   You can use DCLGEN to generate these declarations.
3. Define the items that your program can use to check whether an SQL statement executed successfully. You can either define an SQL communications area (SQLCA) or declare SQLSTATE and SQLCODE host variables.
4. Define at least one SQL descriptor area (SQLDA).
5. Declare any of the following data items for passing data between Db2 and a host language:
   - "Host variables" on page 477
   - "Host-variable arrays" on page 478
   - "Host structures" on page 479

   Ensure that you use the appropriate data types. For details, see "Compatibility of SQL and language data types" on page 484
6. Code SQL statements to access Db2 data. Make sure to delimit the statements correctly for the specific programming language.

   For more information about coding SQL statements in host languages, see the language-specific information for your programming language:
   - Assembler
   - C and C++
   - COBOL
   - Fortran
   - Java
   - ODBC
   - PL/I
   - REXX

   Consider using cursors to select a set of rows and then process the set either one row at a time or one rowset at a time.
7. Check the execution of the SQL statements.
8. Handle any SQL error codes.

What to do next

"Writing applications that enable users to create and modify tables" on page 540
"Saving SQL statements that are translated from user requests" on page 541

Related concepts

Example programs that call stored procedures
Examples can be used as models when you write applications that call stored procedures. In addition, *prefix*.SDSNSAMP contains sample jobs DSNTEJ6P and DSNTEJ6S and programs DSN8EP1 and DSN8EP2, which you can run.

XML data in embedded SQL applications (Db2 Programming for XML)
Introduction to Db2 ODBC (Db2 Programming for ODBC)
JDBC application programming (Db2 Application Programming for Java)
SQLJ application programming (Db2 Application Programming for Java)

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Programming applications for performance (Db2 Performance)

Retrieving a set of rows by using a cursor
In an application program, you can retrieve a set of rows from a table or a result table that is returned by a stored procedure. You can retrieve one or more rows at a time.

Writing efficient SQL queries (Db2 Performance)

Declaring table and view definitions

Before your program issues SQL statements that select, insert, update, or delete data, the program needs to declare the tables and views that those statements access.

About this task

Your program is not required to declare tables or views, but doing so offers the following advantages:

- Clear documentation in the program

  The declaration specifies the structure of the table or view and the data type of each column. You can refer to the declaration for the column names and data types in the table or view.
- Assurance that your program uses the correct column names and data types

  The Db2 precompiler uses your declarations to make sure that you have used correct column names and data types in your SQL statements. The Db2 precompiler issues a warning message when the column names and data types in SQL statements do not correspond to the table and view declarations in your program.

Procedure

To declare table and view definitions, use one of the following methods:

- Include an SQL DECLARE TABLE statement in your program. Specify the name of the table or view and list each column and its data type.

  When you declare a table or view that contains a column with a distinct type, declare that column with the source type of the distinct type rather than with the distinct type itself. When you declare the column with the source type, Db2 can check embedded SQL statements that reference that column at precompile time.

  In a COBOL program, code the DECLARE TABLE statement in the WORKING-STORAGE SECTION or LINKAGE SECTION within the DATA DIVISION.

  For example, the following DECLARE TABLE statement in a COBOL program defines the DSN8B10.DEPT table:

```
EXEC SQL
 DECLARE DSN8B10.DEPT TABLE
   (DEPTNO    CHAR(3)           NOT NULL,
    DEPTNAME  VARCHAR(36)       NOT NULL,
    MGRNO     CHAR(6)                   ,
    ADMRDEPT  CHAR(3)           NOT NULL,
    LOCATION  CHAR(16)                  )
END-EXEC.
```

- Use DCLGEN, the declarations generator that is supplied with Db2, to create these declarations for you and then include them in your program.

  Restriction: You can use DCLGEN for only C, COBOL, and PL/I programs.

Related concepts

DCLGEN (declarations generator)

Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Related reference

DECLARE TABLE (Db2 SQL)
DCLGEN (DECLARATIONS GENERATOR) (DSN) (Db2 Commands)

DCLGEN (declarations generator)

Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

DCLGEN generates a table or view declaration and puts it into a member of a partitioned data set that you can include in your program. When you use DCLGEN to generate a table declaration, Db2 gets the relevant information from the Db2 catalog. The catalog contains information about the table or view definition and the definition of each column within the table or view. DCLGEN uses this information to produce an SQL DECLARE TABLE statement for the table or view and a corresponding PL/I or C structure declaration or COBOL record description.

Related reference

DCLGEN (DECLARATIONS GENERATOR) (DSN) (Db2 Commands)

Generating table and view declarations by using DCLGEN

Your program should declare the tables and views that it accesses. For C, COBOL, and PL/I programs, you can use DCLGEN to produce these declarations, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Before you begin

Requirements:

- Db2 must be active before you can use DCLGEN.
- You can use DCLGEN for table declarations only if the table or view that you are declaring already exists.
- If you use DCLGEN, you must use it before you precompile your program.

Procedure

To generate table and view declarations by using DCLGEN:

1. Invoke DCLGEN by performing one of the following actions:
   - **To start DCLGEN from ISPF through DB2I:** Select the DCLGEN option on the DB2I Primary Option Menu panel. Then follow the detailed instructions for generating table and view declarations by using DCLGEN from DB2I.
   - **To start DCLGEN directly from TSO:** Sign on to TSO, issue the TSO command DSN, and then issue the subcommand DCLGEN.
   - **To start DCLGEN directly from a CLIST:** From a CLIST, running in TSO foreground or background, issue DSN and then DCLGEN.
   - **To start DCLGEN with JCL:** Supply the required information in JCL and run DCLGEN in batch. Use the sample jobs DSNTEJ2C and DSNTEJ2P in the *prefix*.SDSNSAMP library as models.

     Requirement: If you want to start DCLGEN in the foreground and your table names include DBCS characters, you must provide and display double-byte characters. If you do not have a terminal that displays DBCS characters, you can enter DBCS characters by using the hex mode of ISPF edit.

   DCLGEN creates the declarations in the specified data set.

DCLGEN generates a table or column name in the DECLARE statement as a non-delimited identifier unless at least one of the following conditions is true:

- The name contains special characters and is not a DBCS string.
- The name is a DBCS string, and you have requested delimited DBCS names.

2. If you use an SQL reserved word as an identifier, edit the DCLGEN output to add the appropriate SQL delimiters.
3. Make any other necessary edits to the DCLGEN output.

   DCLGEN produces output that is intended to meet the needs of most users, but occasionally, you need to edit the DCLGEN output to work in your specific case. For example, DCLGEN is unable to determine whether a column that is defined as NOT NULL also contains the DEFAULT clause, so you must edit the DCLGEN output to add the DEFAULT clause to the appropriate column definitions.

   DCLGEN produces declarations based on the encoding scheme of the source table. Therefore, if your application uses a different encoding scheme, you might need to manually adjust the declarations. For example, if your source table is in EBCDIC with CHAR columns and your application is in COBOL, DCLGEN produces declarations of type PIC X. However, suppose your host variables in your COBOL application are UTF-16. In this case, you will need to manually change the declarations to be type PIC N USAGE NATIONAL.

Related reference
DCLGEN (DECLARATIONS GENERATOR) (DSN) (Db2 Commands)
DSN (TSO) (Db2 Commands)
Reserved words (Db2 SQL)

Generating table and view declarations by using DCLGEN from DB2I

DCLGEN generates table and view declarations and the corresponding variable declarations for C, COBOL, and PL/I programs so that you do not need to code these statements yourself. The easiest way to start DCLGEN is through DB2I.

Procedure

To generate table and view declarations by using DCLGEN from DB2I:

1. From the DB2I Primary Option Menu panel, select the **DCLGEN** option.

   The following DCLGEN panel is displayed:

```
DSNEDP01                   DCLGEN                             SSID: DSN
===>

Enter table name for which declarations are required:
 1  SOURCE TABLE NAME ===>

 2  TABLE OWNER ..... ===>

 3  AT LOCATION ..... ===>                                      (Optional)
Enter destination data set:          (Can be sequential or partitioned)
 4  DATA SET NAME ... ===>
 5  DATA SET PASSWORD ===>           (If password protected)
Enter options as desired:
 6  ACTION .......... ===> ADD       (ADD new or REPLACE old declaration)
 7  COLUMN LABEL .... ===> NO        (Enter YES for column label)
 8  STRUCTURE NAME .. ===>                                      (Optional)
 9  FIELD NAME PREFIX ===>                                      (Optional)
10  DELIMIT DBCS .... ===> YES       (Enter YES to delimit DBCS identifiers)
11  COLUMN SUFFIX ... ===> NO        (Enter YES to append column name)
12  INDICATOR VARS .. ===> NO        (Enter YES for indicator variables)
13  ADDITIONAL OPTIONS===> YES       (Enter YES to change additional options)

PRESS: ENTER to process    END to exit      HELP for more information
```

   Figure 23. DCLGEN panel

2. Complete the following fields on the DCLGEN panel:

1 SOURCE TABLE NAME
Is the unqualified name of the table, view, or created temporary table for which you want DCLGEN to produce SQL data declarations. The table can be stored at your Db2 location or at another Db2 location. To specify a table name at another Db2 location, enter the table qualifier in the TABLE OWNER field and the location name in the AT LOCATION field. DCLGEN generates a three-part table name from the SOURCE TABLE NAME, TABLE OWNER, and AT LOCATION fields. You can also use an alias for a table name.

To specify a table name that contains special characters or blanks, enclose the name in apostrophes. If the name contains apostrophes, you must double each one(' '). For example, to specify a table named DON'S TABLE, enter the following text:

```
'DON''S TABLE'
```

The underscore is not handled as a special character in DCLGEN. For example, the table name JUNE_PROFITS does not need to be enclosed in apostrophes. Because COBOL field names cannot contain underscores, DCLGEN substitutes hyphens (-) for single-byte underscores in COBOL field names that are built from the table name.

You do not need to enclose DBCS table names in apostrophes.

If you do not enclose the table name in apostrophes, Db2 converts lowercase characters to uppercase.

2 TABLE OWNER
Is the schema qualifier of the source table. If you do not specify this value and the table is a local table, Db2 assumes that the table qualifier is your TSO logon ID. If the table is at a remote location, you must specify this value.

3 AT LOCATION
Is the location of a table or view at another Db2 subsystem. The value of the AT LOCATION field becomes a prefix for the table name on the SQL DECLARE statement, as follows: *location_name*, *schema_name*, *table_name* For example, if the location name is PLAINS_GA, the schema name is CARTER, and the table name is CROP_YIELD_89, the following table name is included in the SQL DECLARE statement: PLAINS_GA.CARTER.CROP_YIELD_89

The default is the local location name. This field applies to Db2 private protocol access only. The location must be another Db2 for z/OS subsystem.

4 DATA SET NAME
Is the name of the data set that you allocated to contain the declarations that DCLGEN produces. You must supply a name; no default exists.

The data set must already exist and be accessible to DCLGEN. The data set can be either sequential or partitioned. If you do not enclose the data set name in apostrophes, DCLGEN adds a standard TSO prefix (user ID) and suffix (language). DCLGEN determines the host language from the DB2I defaults panel.

For example, for library name LIBNAME(MEMBNAME), the name becomes *userid*.libname.*language*(membname) For library name LIBNAME, the name becomes *userid*.libname.*language*.

If this data set is password protected, you must supply the password in the DATA SET PASSWORD field.

5 DATA SET PASSWORD
Is the password for the data set that is specified in the DATA SET NAME field, if the data set is password protected. The password is not displayed on your terminal, and it is not recognized if you issued it from a previous session.

6 ACTION
Specifies what DCLGEN is to do with the output when it is sent to a partitioned data set. (The option is ignored if the data set you specify in the DATA SET NAME field is sequential.) You can specify one of the following values:

ADD
Indicates that an old version of the output does not exist and creates a new member with the specified data set name. ADD is the default.

REPLACE
Replaces an old version, if it already exists. If the member does not exist, this option creates a new member.

7 COLUMN LABEL
Specifies whether DCLGEN is to include labels that are declared on any columns of the table or view as comments in the data declarations. (The SQL LABEL statement creates column labels to use as supplements to column names.) You can specify one of the following values:

YES
Include column labels.

NO
Ignore column labels. NO is the default.

8 STRUCTURE NAME
Is the name of the generated data structure. The name can be up to 31 characters. If the name is not a DBCS string, and the first character is not alphabetic, enclose the name in apostrophes. If you use special characters, be careful to avoid name conflicts.

If you leave this field blank, DCLGEN generates a name that contains the table or view name with a prefix of DCL. If the language is COBOL or PL/I and the table or view name consists of a DBCS string, the prefix consists of DBCS characters.

For C, lowercase characters that you enter in this field are not converted to uppercase.

9 FIELD NAME PREFIX
Specifies a prefix that DCLGEN uses to form field names in the output. For example, if you choose ABCDE, the field names generated are ABCDE1, ABCDE2, and so on.

You can specify a field name prefix of up to 28 bytes that can include special and double-byte characters. If you specify a single-byte or mixed-string prefix and the first character is not alphabetic, enclose the prefix in apostrophes. If you use special characters, be careful to avoid name conflicts.

For COBOL and PL/I, if the name is a DBCS string, DCLGEN generates DBCS equivalents of the suffix numbers.

For C, lowercase characters that you enter in this field do not converted to uppercase.

If you leave this field blank, the field names are the same as the column names in the table or view.

10 DELIMIT DBCS
Specifies whether DCLGEN is to delimit DBCS table names and column names in the table declaration. You can specify one of the following values:

YES
Specifies that DCLGEN is to enclose the DBCS table and column names with SQL delimiters.

NO
Specifies that DCLGEN is not to delimit the DBCS table and column names.

11 COLUMN SUFFIX
Specifies whether DCLGEN is to form field names by attaching the column name as a suffix to the value that you specify in FIELD NAME PREFIX. You can specify one of the following values:

YES
Specifies that DCLGEN is to use the column name as a suffix. For example, if you specify YES, the field name prefix is NEW, and the column name is EMPNO, the field name is NEWEMPNO.

If you specify YES, you must also enter a value in FIELD NAME PREFIX. If you do not enter a field name prefix, DCLGEN issues a warning message and uses the column names as the field names.

NO
Specifies that DCLGEN is not to use the column name as a suffix. The default is NO.

12 INDICATOR VARS
Specifies whether DCLGEN is to generate an array of indicator variables for the host variable structure. You can specify one of the following values:

YES
Specifies that DCLGEN is to generate an array of indicator variables for the host variable structure.

If you specify YES, the array name is the table name with a prefix of I (or DBCS letter <I> if the table name consists solely of double-byte characters). The form of the data declaration depends on the language, as shown in the following table. *n* is the number of columns in the table.

Table 86. Declarations for indicator variable arrays from DCLGEN

Language	Declaration form
C	`short int I`*`table-name`*`[`*`n`*`];`
COBOL	`01 I`*`table-name`*` PIC S9(4) USAGE COMP OCCURS `*`n`*` TIMES.`
PL/I	`DCL I`*`table-name`*`(`*`n`*`) BIN FIXED(15);`

For example, suppose that you define the following table:

```
CREATE TABLE HASNULLS (CHARCOL1 CHAR(1), CHARCOL2 CHAR(1));
```

If you request an array of indicator variables for a COBOL program, DCLGEN might generate the following host variable declaration:

```
01  DCLHASNULLS.
    10 CHARCOL1             PIC X(1).
    10 CHARCOL2             PIC X(1).
01  IHASNULLS PIC S9(4) USAGE COMP OCCURS 2 TIMES.
```

NO
Specifies that DCLGEN is not to generate an array of indicator variables. The default is NO.

13 ADDITIONAL OPTIONS
Indicates whether to display the panel for additional DCLGEN options, including the break point for statement tokens and whether to generate DECLARE VARIABLE statements for FOR BIT DATA columns. You can specify YES or NO. The default is YES.

If you specified YES in the ADDITIONAL OPTIONS field, the following ADDITIONAL DCLGEN OPTIONS panel is displayed:

```
DSNEDP02             ADDITIONAL DCLGEN OPTIONS                SSID: DSN
===>

Enter options as desired:
 1  RIGHT MARGIN .... ===> 72        (Enter 72 or 80)

 2  FOR BIT DATA .... ===> NO        (Enter YES to declare SQL variables for
                                      FOR BIT DATA columns)

PRESS: ENTER to process    END to exit      HELP for more information
```

Figure 24. ADDITIONAL DCLGEN OPTIONS panel

Otherwise, DCLGEN creates the declarations in the specified data set.

3. If the ADDITIONAL DCLGEN OPTIONS panel is displayed, complete the following fields on that panel:

1 RIGHT MARGIN
Specifies the break point for statement tokens that must be wrapped to one or more subsequent records. You can specify column 72 or column 80.

The default is 72.

2 FOR BIT DATA
Specifies whether DCLGEN is to generate a DECLARE VARIABLE statement for SQL variables for columns that are declared as FOR BIT DATA. This statement is required in Db2 applications that meet all of the following criteria:

- are written in COBOL
- have host variables for FOR BIT DATA columns
- are prepared with the SQLCCSID option of the Db2 coprocessor.

You can specify YES or NO. The default is NO.

If the table or view does not have FOR BIT DATA columns, DCLGEN does not generate this statement.

DCLGEN creates the declarations in the specified data set.

Related reference
The DB2I primary option menu (Introduction to Db2 for z/OS)
LABEL (Db2 SQL)

Data types that DCLGEN uses for variable declarations

DCLGEN produces declarations for tables and views and the corresponding host variable structures for C, COBOL, and PL/I programs. DCLGEN derives the variable names and data types for these declarations based on the source tables in the database.

The following table lists the C, COBOL, and PL/I data types that DCLGEN uses for variable declarations based on the corresponding SQL data types that are used in the source tables. *var* represents a variable name that DCLGEN provides.

Table 87. Type declarations that DCLGEN generates

SQL data type[1]	**C**	**COBOL**	**PL/I**
SMALLINT	`short int`	`PIC S9(4) USAGE COMP`	`BIN FIXED(15)`
INTEGER	`long int`	`PIC S9(9) USAGE COMP`	`BIN FIXED(31)`
DECIMAL(p,s) or NUMERIC(p,s)	`decimal(p,s)`[2]	`PIC S9(p-s)V9(s) USAGE COMP-3`	`DEC FIXED(p,s)` If p>15, the PL/I compiler must support this precision, or a warning is generated.
REAL or FLOAT(n) 1 <= n <= 21	`float`	`USAGE COMP-1`	`BIN FLOAT(n)`
DOUBLE PRECISION, DOUBLE, or FLOAT(n)	`double`	`USAGE COMP-2`	`BIN FLOAT(n)`
CHAR(1)	`char`	`PIC X(1)`	`CHAR(1)`
CHAR(n)	`char` *`var`* `[n+1]`	`PIC X(n)`	`CHAR(n)`

Table 87. Type declarations that DCLGEN generates (continued)

SQL data type[1]	C	COBOL	PL/I
VARCHAR(n)	struct {short int *var*_len; char *var*_data[n]; } *var*;	10 *var*. 49 *var*_LEN PIC 9(4) USAGE COMP. 49 *var*_TEXT PIC X(n).	CHAR(n) VAR
CLOB(n)[3]	SQL TYPE IS CLOB_LOCATOR	USAGE SQL TYPE IS CLOB-LOCATOR	SQL TYPE IS CLOB_LOCATOR
GRAPHIC(1)	sqldbchar	PIC G(1)	GRAPHIC(1)
GRAPHIC(n) n > 1	sqldbchar *var*[n+1];	PIC G(n) USAGE DISPLAY-1.[4] or PIC N(n).[4]	GRAPHIC(n)
VARGRAPHIC(n)	struct VARGRAPH {short len; sqldbchar data[n]; } *var*;	10 *var*. 49 *var*_LEN PIC 9(4) USAGE COMP. 49 *var*_TEXT PIC G(n) USAGE DISPLAY-1.[4] or 10 *var*. 49 *var*_LEN PIC 9(4) USAGE COMP. 49 *var*_TEXT PIC N(n).[4]	GRAPHIC(n) VAR
DBCLOB(n)[3]	SQL TYPE IS DBCLOB_LOCATOR	USAGE SQL TYPE IS DBCLOB-LOCATOR	SQL TYPE IS DBCLOB_LOCATOR
BINARY(*n*)	SQL TYPE IS BINARY(*n*)	USAGE SQL TYPE IS BINARY(*n*)	SQL TYPE IS BINARY(*n*)
VARBINARY(*n*)	SQL TYPE IS VARBINARY(*n*)	USAGE SQL TYPE IS VARBINARY(*n*)	SQL TYPE IS VARBINARY(*n*)
BLOB(n)[3]	SQL TYPE IS BLOB_LOCATOR	USAGE SQL TYPE IS BLOB-LOCATOR	SQL TYPE IS BLOB_LOCATOR
DATE	char *var*[11][5]	PIC X(10)[5]	CHAR(10)[5]
TIME	char *var*[9][6]	PIC X(8)[6]	CHAR(8)[6]
TIMESTAMP	char *var*[27]	PIC X(26)	CHAR(26)
TIMESTAMP(0)	char *var*[20]	PIC X(19)	CHAR(19)
TIMESTAMP(*p*) *p* > 0	char *var*[21+*p*]	PIC X(20+*p*)	CHAR(20+*p*)
TIMESTAMP(0) WITH TIME ZONE	struct {short int *var*_len; char *var*_data[147]; } *var*;	01 *var*. 49 *var*_LEN PIC S9(4) COMP. 49 *var*_TEXT PIC X(147).	DCL var CHAR(147) VAR;

Table 87. Type declarations that DCLGEN generates (continued)

SQL data type[1]	C	COBOL	PL/I
TIMESTAMP(*p*) WITH TIME ZONE	`struct {short int var_len; char var_data[148 + p]; } var;`	`01 var. 49 var_LEN PIC S9(4) COMP. 49 var_TEXT PIC X(148 + p).`	`DCL var CHAR(148 + p) VAR;`
ROWID	`SQL TYPE IS ROWID`	`USAGE SQL TYPE IS ROWID`	`SQL TYPE IS ROWID`
BIGINT	`long long int`	`PIC S9(18) USAGE COMP`	`FIXED BIN(63)`
XML[7]	`SQL TYPE IS XML AS CLOB(1M)`	`SQL TYPE IS XML AS CLOB(1M)`	`SQL TYPE IS XML AS CLOB(1M)`

Notes:

1. For a distinct type, DCLGEN generates the host language equivalent of the source data type.
2. If your C compiler does not support the decimal data type, edit your DCLGEN output and replace the decimal data declarations with declarations of type double.
3. For a BLOB, CLOB, or DBCLOB data type, DCLGEN generates a LOB locator.
4. DCLGEN chooses the format based on the character that you specify as the DBCS symbol on the COBOL Defaults panel.
5. This declaration is used unless a date installation exit routine exists for formatting dates, in which case the length is that specified for the LOCAL DATE LENGTH installation option.
6. This declaration is used unless a time installation exit routine exists for formatting times, in which case the length is that specified for the LOCAL TIME LENGTH installation option.
7. The default setting for XML is 1M; however, you might need to adjust it.

Including declarations from DCLGEN in your program

After you use DCLGEN to produce declarations for tables, views, and variables for your C, COBOL, or PL/I program, you should include these declarations in your program.

Before you begin

Recommendation: To ensure that your program uses a current description of the table, use DCLGEN to generate the table's declaration and store it as a member in a library (usually a partitioned data set) just before you precompile the program.

Procedure

Code the following SQL INCLUDE statement in your program:

```
EXEC SQL
  INCLUDE member-name
END-EXEC.
```

member-name is the name of the data set member where the DCLGEN output is stored.

Example

Suppose that you used DCLGEN to generate a table declaration and corresponding COBOL record description for the table DSN8B10.EMP, and those declarations were stored in the data set member DECEMP. (A COBOL record description is a two-level host structure that corresponds to the columns of a table's row.) To include those declarations in your program, include the following statement in your COBOL program:

```
EXEC SQL
  INCLUDE DECEMP
END-EXEC.
```

Related reference

INCLUDE (Db2 SQL)

Example: Adding DCLGEN declarations to a library

You can use DCLGEN to generate table and variable declarations for C, COBOL, and PL/I programs. If you store these declarations in a library, you can later integrate them into your program with a single SQL INCLUDE statement.

This example adds a table declaration and a corresponding host-variable structure to a library. This example is based on the following scenario:

- The library name is *prefix*.TEMP.COBOL.
- The member is a new member named VPHONE.
- The table is a local table named DSN8B10.VPHONE.
- The host-variable structure is for COBOL.
- The structure receives the default name DCLVPHONE.

Throughout this example, information that you must enter on each panel is in bold-faced type.

In this scenario, to add a table declaration and a corresponding host variable structure for DSN8B10.VPHONE to the library *prefix*.TEMP.COBOL, complete the following steps:

1. Specify COBOL as the host language by completing the following actions:
   a. On the ISPF/PDF menu, select option **D** to display the DB2I DEFAULTS PANEL l panel.
   b. Specify IBMCOB as the application language, as shown in the following figure and press Enter.

```
DSNEOP01                   DB2I DEFAULTS PANEL 1
COMMAND ===>_

Change defaults as desired:

 1  DB2 NAME ............. ===> DSN       (Subsystem identifier)
 2  DB2 CONNECTION RETRIES ===> 0         (How many retries for DB2 connection)
 3  APPLICATION LANGUAGE   ===> IBMCOB     (ASM, C, CPP, IBMCOB, FORTRAN, PLI)
 4  LINES/PAGE OF LISTING  ===> 80        (A number from 5 to 999)
 5  MESSAGE LEVEL ........ ===> I         (Information, Warning, Error, Severe)
 6  SQL STRING DELIMITER   ===> DEFAULT   (DEFAULT, ' or ")
 7  DECIMAL POINT ........ ===> .         (. or ,)
 8  STOP IF RETURN CODE >= ===> 8         (Lowest terminating return code)
 9  NUMBER OF ROWS         ===> 20        (For ISPF Tables)
10  CHANGE HELP BOOK NAMES?===> NO        (YES to change HELP data set names)
11  AS USER                ===>           (Userid to associate with the trusted
                                           connection)

PRESS: ENTER to process        END to cancel     HELP for more information
```

   Figure 25. DB2I defaults panel—changing the application language

   The DB2I DEFAULTS PANEL 2 panel for COBOL is then displayed.

   c. Complete the DB2I DEFAULTS PANEL 2 panel, shown in the following figure, as needed and press Enter to save the new defaults, if any.

```
DSNEOP02                     DB2I DEFAULTS PANEL 2
COMMAND ===>_

Change defaults as desired:

1  DB2I JOB STATEMENT:   (Optional if your site has a SUBMIT exit)
   ===> //ADMF001A JOB (ACCOUNT),'NAME'
   ===> //*
   ===> //*
   ===> //*


   COBOL DEFAULTS:                       (For IBMCOB)
2  COBOL STRING DELIMITER ===> DEFAULT   (DEFAULT, ' or ")
3  DBCS SYMBOL FOR DCLGEN ===> G         (G/N - Character in PIC clause)
```

Figure 26. The COBOL defaults panel

The DB2I Primary Option menu is displayed.

2. Generate the table and host structure declarations by completing the following actions:

   a. On the DB2I Primary Option menu, select the **DCLGEN** option and press Enter to display the DCLGEN panel.

   b. Complete the fields as shown in the following figure and press Enter.

```
DSNEDP01                   DCLGEN                            SSID: DSN
===>

Enter table name for which declarations are required:
 1  SOURCE TABLE NAME ===>
DSN8B10.VPHONE

 2  TABLE OWNER ..... ===>

 3  AT LOCATION ..... ===>                                    (Optional)
Enter destination data set:           (Can be sequential or partitioned)
 4  DATA SET NAME ... ===>
TEMP(VPHONEC)
 5  DATA SET PASSWORD ===>            (If password protected)
Enter options as desired:
 6  ACTION .......... ===> ADD       (ADD new or REPLACE old declaration)
 7  COLUMN LABEL .... ===> NO         (Enter YES for column label)
 8  STRUCTURE NAME .. ===>                                    (Optional)
 9  FIELD NAME PREFIX ===>                                    (Optional)
10  DELIMIT DBCS .... ===> YES        (Enter YES to delimit DBCS identifiers)
11  COLUMN SUFFIX ... ===> NO        (Enter YES to append column name)
12  INDICATOR VARS .. ===> NO         (Enter YES for indicator variables)
13  ADDITIONAL OPTIONS===> NO         (Enter YES to change additional options)

PRESS: ENTER to process    END to exit      HELP for more information
```

Figure 27. DCLGEN panel—selecting source table and destination data set

A successful completion message, such as the one in the following figure, is displayed at the top of your screen.

```
DSNE905I EXECUTION COMPLETE, MEMBER VPHONEC ADDED
***
```

Figure 28. Successful completion message

Db2 again displays the DCLGEN screen, as shown in the following figure.

```
 DSNEDP01                   DCLGEN                             SSID: DSN
===>

Enter table name for which declarations are required:
 1  SOURCE TABLE NAME ===>
DSN8B10.VPHONE

 2  TABLE OWNER ..... ===>

 3  AT LOCATION ..... ===>                                      (Optional)
Enter destination data set:          (Can be sequential or partitioned)
 4  DATA SET NAME ... ===> TEMP(VPHONEC)
 5  DATA SET PASSWORD ===>           (If password protected)
Enter options as desired:
 6  ACTION .......... ===> ADD      (ADD new or REPLACE old declaration)
 7  COLUMN LABEL .... ===> NO        (Enter YES for column label)
 8  STRUCTURE NAME .. ===>                                      (Optional)
 9  FIELD NAME PREFIX ===>                                      (Optional)
10  DELIMIT DBCS .... ===> YES       (Enter YES to delimit DBCS identifiers)
11  COLUMN SUFFIX ... ===> NO        (Enter YES to append column name)
12  INDICATOR VARS .. ===> NO        (Enter YES for indicator variables)
13  ADDITIONAL OPTIONS===> NO        (Enter YES to change additional options)

PRESS: ENTER to process    END to exit      HELP for more information
```

Figure 29. DCLGEN panel—displaying system and user return codes

c. Press Enter to return to the DB2I Primary Option menu.

3. Exit from DB2I.
4. Examine the DCLGEN output by selecting either the browse or the edit option from the ISPF/PDF menu to view the results in the specified data set member.

   For this example, the data set to edit is *prefix*.TEMP.COBOL(VPHONEC). This data set member contains the following information.

```
***** DCLGEN TABLE(DSN8B10.VPHONE)                                 ***
*****        LIBRARY(SYSADM.TEMP.COBOL(VPHONEC))                   ***
*****        QUOTE                                                 ***
***** ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS ***
         EXEC SQL DECLARE DSN8B10.VPHONE TABLE
         ( LASTNAME                       VARCHAR(15) NOT NULL,
           FIRSTNAME                      VARCHAR(12) NOT NULL,
           MIDDLEINITIAL                  CHAR(1) NOT NULL,
           PHONENUMBER                    VARCHAR(4) NOT NULL,
           EMPLOYEENUMBER                 CHAR(6) NOT NULL,
           DEPTNUMBER                     CHAR(3) NOT NULL,
           DEPTNAME                       VARCHAR(36) NOT NULL
         ) END-EXEC.
***** COBOL DECLARATION FOR TABLE DSN8B10.VPHONE                ******
     01  DCLVPHONE.
         10 LASTNAME.
            49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
            49 LASTNAME-TEXT     PIC X(15).
         10 FIRSTNAME.
            49 FIRSTNAME-LEN     PIC S9(4) USAGE COMP.
            49 FIRSTNAME-TEXT    PIC X(12).
         10 MIDDLEINITIAL        PIC X(1).
         10 PHONENUMBER.
            49 PHONENUMBER-LEN   PIC S9(4) USAGE COMP.
            49 PHONENUMBER-TEXT  PIC X(4).
         10 EMPLOYEENUMBER       PIC X(6).
         10 DEPTNUMBER           PIC X(3).
         10 DEPTNAME.
            49 DEPTNAME-LEN      PIC S9(4) USAGE COMP.
            49 DEPTNAME-TEXT     PIC X(36).
***** THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 7  ******
```

   You can now pull these declarations into your program by using an SQL INCLUDE statement.

Defining the items that your program can use to check whether an SQL statement executed successfully

If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

About this task

Whether you define the SQLCODE or SQLSTATE variables or an SQLCA in your program depends on what you specify for the SQL processing option STDSQL.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Related tasks

Defining the SQL communications area, SQLSTATE, and SQLCODE in assembler
Assembler programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in C and C++
C and C++ programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL
COBOL programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in Fortran
Fortran programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I
PL/I programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX
When Db2 prepares a REXX program that contains SQL statements, Db2 automatically includes an SQLCA in the program.

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Description of SQLCA fields (Db2 SQL)

INCLUDE (Db2 SQL)

The REXX SQLCA (Db2 SQL)

Defining SQL descriptor areas (SQLDA)

If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

About this task

If your program includes any of the following statement variations, you must include an SQLDA in your program:

- CALL ... USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE CURSOR *host-variable* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE PROCEDURE *host-variable* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- EXECUTE ... USING DESCRIPTOR *descriptor-name*
- FETCH ... INTO DESCRIPTOR *descriptor-name*
- OPEN ... USING DESCRIPTOR *descriptor-name*
- PREPARE ... INTO *descriptor-name*

Unlike the SQLCA, a program can have more than one SQLDA, and an SQLDA can have any valid name.

Procedure

Take the actions that are appropriate for the programming language that you use:

- "Defining SQL descriptor areas (SQLDA) in assembler" on page 556
- "Defining SQL descriptor areas (SQLDA) in C and C++" on page 583
- "Defining SQL descriptor areas (SQLDA) in COBOL" on page 649
- "Defining SQL descriptor areas in (SQLDA) Fortran" on page 687
- "Defining SQL descriptor areas (SQLDA) in PL/I" on page 705
- "Defining SQL descriptor areas (SQLDA) in REXX" on page 743

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Description of SQLCA fields (Db2 SQL)
The REXX SQLCA (Db2 SQL)

Declaring host variables and indicator variables

You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Procedure

Use the techniques that are appropriate for the programming language that you use.

Related tasks

Accessing data by using a rowset-positioned cursor
A rowset-positioned cursor is a cursor that can return one or more rows for a single fetch operation. The cursor is positioned on the set of rows that are to be fetched.

Determining whether a retrieved value in a host variable is null or truncated
Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Host variables

Use host variables to pass a single data item between Db2 and your application.

A *host variable* is a single data item that is declared in the host language to be used within an SQL statement. You can use host variables in application programs that are written in the following languages: assembler, C, C++, COBOL, Fortran, and PL/I to perform the following actions:

- Retrieve data into the host variable for your application program's use
- Place data into the host variable to insert into a table or to change the contents of a row
- Use the data in the host variable when evaluating a WHERE or HAVING clause
- Assign the value that is in the host variable to a special register, such as CURRENT SQLID and CURRENT DEGREE
- Insert null values into columns by using a host indicator variable that contains a negative value
- Use the data in the host variable in statements that process dynamic SQL, such as EXECUTE, PREPARE, and OPEN

Related concepts

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Related reference

Host variables in assembler
In assembler programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Host variables in C and C++
In C and C++ programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Host variables in COBOL
In COBOL programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set and table locators and LOB and XML file reference variables.

Host variables in Fortran
In Fortran programs, you can specify numeric, character, LOB, and ROWID host variables. You can also specify result set and LOB locators.

Host variables in PL/I

In PL/I programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Host-variable arrays

You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

You can use host-variable arrays for the following actions:

- Retrieve data into host-variable arrays for your application use by your application
- Place data into host-variable arrays to insert rows into a table
- Retrieve data for the source of a merge operation.

Host-variable arrays can be referenced only as a simple reference in the following contexts. In syntax diagrams, *host-variable-array* designates a reference to a host-variable array.

- In a FETCH statement for a multiple-row fetch. See FETCH (Db2 SQL).
- In an INSERT statement with multiple rows of source data. This is also referred to as multiple-row insert. See INSERT (Db2 SQL).
- In a MERGE statement with multiple rows of source data. See MERGE (Db2 SQL).
- In an EXECUTE statement to provide a value for a parameter marker in a dynamic multi-row INSERT or MERGE. See EXECUTE (Db2 SQL).

Host-variable arrays are defined by statements of the host language, as explained in the following topics:

- "Host-variable arrays in C and C++" on page 595
- "Host-variable arrays in COBOL" on page 660
- "Host-variable arrays in PL/I" on page 712

Tip: Host-variable arrays are not supported for assembler, FORTRAN, or REXX programs. However, you can use SQL descriptor areas (SQLDA) to achieve similar results in any host language. For more information see "Defining SQL descriptor areas (SQLDA)" on page 476.

Example

GUPI

The following statement uses the main host-variable array, COL1, and the corresponding indicator array, COL1IND. Assume that COL1 has 10 elements. The first element in the array corresponds to the first value, and so on. COL1IND must have at least 10 entries.

```
EXEC SQL
  SQL FETCH FIRST ROWSET FROM C1 FOR 5 ROWS
    INTO :COL1 :COL1IND
END-EXEC.
```

GUPI

Related concepts

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

Related tasks

Inserting multiple rows of data from host-variable arrays

Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Related reference

Host-variable arrays in C and C++
In C and C++ programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host-variable arrays in COBOL
In COBOL programs, you can specify numeric, character, graphic, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host-variable arrays in PL/I
In PL/I programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host structures

Use host structures to pass a group of host variables between Db2 and your application.

A *host structure* is a group of host variables that can be referenced with a single name. You can use host structures in all host languages except REXX. You define host structures with statements in the host language. You can refer to a host structure in any context where you want to refer to the list of host variables in the structure. A host structure reference is equivalent to a reference to each of the host variables within the structure in the order in which they are defined in the structure declaration. You can also use indicator variables (or indicator structures) with host structures.

Related tasks

Retrieving a single row of data into a host structure
If you know that your query returns multiple column values for only one row, you can specify a host structure to contain the column values.

Related reference

Host structures in C and C++
A C host structure contains an ordered group of data fields.

Host structures in COBOL
A COBOL host structure is a named set of host variables that are defined in your program's WORKING-STORAGE SECTION or LINKAGE SECTION.

Host structures in PL/I
A PL/I host structure is a structure that contains subordinate levels of scalars. You can use the name of the structure as shorthand notation to reference the list of scalars.

Indicator variables, arrays, and structures

An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

You can use indicator variables to perform the following actions:

- Determine whether the value of an associated output host variable is null or indicate that the value of an input host variable is null
- Determine the original length of a character string that was truncated when it was assigned to a host variable
- Determine that a character value could not be converted when it was assigned to a host variable

- Determine the seconds portion of a time value that was truncated when it was assigned to a host variable
- Indicate that the target column of the host variable is to be set to its defined DEFAULT value, or that the host variable's value is UNASSIGNED and its target column is to be treated as if it had not appeared in the statement.

You can use indicator variable arrays and indicator structures to perform these same actions for individual items in host data arrays and structures.

If you provide an indicator variable for the variable X, when Db2 retrieves a null value for X, it puts a negative value in the indicator variable and does not update X. Your program should check the indicator variable before using X. If the indicator variable is negative, you know that X is null and any value that you find in X is irrelevant. When your program uses variable X to assign a null value to a column, the program should set the indicator variable to a negative number. Db2 then assigns a null value to the column and ignores any value in X.

An indicator variable array contains a series of small integers to help you determine the associated information for the corresponding item in a host data array. When you retrieve data into a host-variable array, you can check the values in the associated indicator array to determine how to handle each data item. If a value in the associated indicator array is negative, you can disregard the contents of the corresponding element in the host-variable array. Values in indicator arrays have the following meanings:

On output to the application, the normal indicator variable can contain the following values:

0
: A 0 (zero), or positive value of the indicator variable specifies that the first host-identifier provides the value of this host variable reference.

-1
: A -1 value indicates that the value that was selected was the null value.

-2
: A -2 value of the indicator variable indicates that a numeric conversion error (such as a divide by 0 or overflow) has occurred. Or indicates a null result because of character string conversion warnings.

-3
: A -3 value of the indicator variable indicates that no value was returned. A -3 value of the indicator variable can also indicate a null result because the cursor's current row is on a hole that was detected during a multiple row FETCH.

positive integer
: If the indicator variable contains a positive integer, the retrieved value is truncated, and the integer is the original length of the string.

positive integer
: The seconds portion of a time if the time is truncated on assignment to a host variable.

On input to Db2, normal indicator variables or extended indicator variables can contain the following values:

0, or positive integer
: Specifies a non-null value. A 0 (zero), or positive value of the indicator variable specifies that the first host-identifier provides the value of this host variable reference.

-1, -2, -3, -4, -6
: Specifies a null value.

-5
- If the extended indicator variable is not enabled, a -5 value specifies the NULL value.
- If the extended indicator variable is enabled, a -5 value specifies the DEFAULT value. A -5 value specifies that the target column for this host variable is to be set to its DEFAULT value.

-7
- If the extended indicator variable is not enabled, a -7 value specifies the NULL value.

- If the extended indicator variable is enabled, a -7 value specifies the an UNASSIGNED value. A -7 value specifies that the target column for this host variable is to be treated as if it hadn't been specified in the statement.

An *indicator structure* is an array of halfword integer variables that supports a specified host structure. If the column values that your program retrieves into a host structure can be null, you can attach an indicator structure name to the host structure name. This name enables Db2 to notify your program about each null value it returns to a host variable in the host structure.

Related concepts

Holes in the result table of a scrollable cursor
A hole in the result table means that the result table does not shrink to fill the space of deleted rows. It also does not shrink to fill the space of rows that have been updated and no longer satisfy the search condition. You cannot access a delete or update hole. However, you can remove holes in specific situations.

Related tasks

Executing SQL statements by using a rowset cursor
You can use rowset cursors to execute multiple-row FETCH statements, positioned UPDATE statements, and positioned DELETE statements.

Related reference

Indicator variables in assembler
An indicator variable is a 2-byte integer (DS HL2). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

Indicator variables, indicator arrays, and host structure indicator arrays in C and C++
An indicator variable is a 2-byte integer (short int). An indicator variable array is an array of 2-byte integers (short int). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

Indicator variables, indicator arrays, and host structure indicator arrays in COBOL
A COBOL indicator variable is a 2-byte binary integer. A COBOL indicator variable array is an array in which each element is declared as a 2-byte binary integer. You can use indicator variable arrays to support COBOL host structures.

Indicator variables in Fortran
An indicator variable is a 2-byte integer (INTEGER*2). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

Indicator variables in PL/I
An indicator variable is a 2-byte integer (or an integer declared as BIN FIXED(15)). An indicator variable array is an array of 2-byte integers. You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

Setting the CCSID for host variables

All Db2 string data, other than binary data, has an encoding scheme and a coded character set ID (CCSID) associated with it. You can associate an encoding scheme and a CCSID with individual host variables. Any data in those host variable is then associated with that encoding scheme and CCSID.

Procedure

Specify the DECLARE VARIABLE statement after the corresponding host variable declaration and before your first reference to that host variable.

This statement associates an encoding scheme and a CCSID with individual host variables. You can use this statement in static or dynamic SQL applications.

Restriction: You cannot use the DECLARE VARIABLE statement to control the CCSID and encoding scheme of data that you retrieve or update by using an SQLDA.

The DECLARE VARIABLE statement has the following effects on a host variable:

- When you use the host variable to update a table, the local subsystem or the remote server assumes that the data in the host variable is encoded with the CCSID and encoding scheme that the DECLARE VARIABLE statement assigns.
- When you retrieve data from a local or remote table into the host variable, the retrieved data is converted to the CCSID and encoding scheme that are assigned by the DECLARE VARIABLE statement.

Example

Suppose that you are writing a C program that runs on a Db2 for z/OS subsystem. The subsystem has an EBCDIC application encoding scheme. The C program retrieves data from the following columns of a local table that is defined with the CCSID UNICODE option:

```
PARTNUM CHAR(10)
JPNNAME GRAPHIC(10)
ENGNAME VARCHAR(30)
```

Because the application encoding scheme for the subsystem is EBCDIC, the retrieved data is EBCDIC. To make the retrieved data Unicode, use DECLARE VARIABLE statements to specify that the data that is retrieved from these columns is encoded in the default Unicode CCSIDs for the subsystem.

Suppose that you want to retrieve the character data in Unicode CCSID 1208 and the graphic data in Unicode CCSID 1200. Use the following DECLARE VARIABLE statements:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvpartnum[11];
EXEC SQL DECLARE :hvpartnum VARIABLE CCSID 1208;
sqldbchar hvjpnname[11];
EXEC SQL DECLARE :hvjpnname VARIABLE CCSID 1200;
struct {
        short len;
        char d[30];
       } hvengname;
EXEC SQL DECLARE :hvengname VARIABLE CCSID 1208;
EXEC SQL END DECLARE SECTION;
```

Related reference

DECLARE VARIABLE (Db2 SQL)

Determining what caused an error when retrieving data into a host variable

Errors that occur when Db2 passes data to host variables in an application are usually caused by a problem in converting from one data type to another. These errors do not affect the position of the cursor.

About this task

For example, suppose that you fetch an integer value of 32768 into a host variable of type SMALLINT. The conversion might cause an error if you do not provide sufficient conversion information to Db2.

The variable to which Db2 assigns the data is called the *output host variable*. If you provide an indicator variable for the output host variable or if data type conversion is not required, Db2 returns a positive SQLCODE for the row in most cases. In other cases where data conversion problems occur, Db2 returns a negative SQLCODE for that row. Regardless of the SQLCODE for the row, no new values are assigned to the host variable or to subsequent variables for that row. Any values that are already assigned to variables remain assigned. Even when a negative SQLCODE is returned for a row, statement processing continues and Db2 returns a positive SQLCODE for the statement (SQLSTATE 01668, SQLCODE +354).

Procedure

To determine what caused an error when retrieving data into a host variable:

1. When Db2 returns SQLCODE = +354, use the GET DIAGNOSTICS statement with the NUMBER option to determine the number of errors and warnings.
   For example, suppose that no indicator variables are provided for the values that are returned by the following statement:

```
FETCH FIRST ROWSET FROM C1 FOR 10 ROWS INTO :hva_col1, :hva_col2;
```

For each row with an error, Db2 records a negative SQLCODE and continues processing until the 10 rows are fetched. When SQLCODE = +354 is returned for the statement, you can use the GET DIAGNOSTICS statement to determine which errors occurred for which rows. The following statement returns num_rows = 10 and num_cond = 3:

```
GET DIAGNOSTICS :num_rows = ROW_COUNT, :num_cond = NUMBER;
```

2. To investigate the errors and warnings, use additional GET DIAGNOSTIC statements with the CONDITION option.
For example, to investigate the three conditions that were reported in the example in the previous step, use the following statements:

Table 88. GET DIAGNOSTIC statements to investigate conditions

Statement	Output
`GET DIAGNOSTICS CONDITION 3 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;`	`sqlstate = 22003` `sqlcode = -304` `row_num = 5`
`GET DIAGNOSTICS CONDITION 2 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;`	`sqlstate = 22003` `sqlcode = -802` `row_num = 7`
`GET DIAGNOSTICS CONDITION 1 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;`	`sqlstate = 01668` `sqlcode = +354` `row_num = 0`

This output shows that the fifth row has a data mapping error (-304) for column 1 and that the seventh row has a data mapping error (-802) for column 2. These rows do not contain valid data, and they should not be used.

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Related information

+354 (Db2 Codes)

Accessing an application defaults module

If your application program currently uses LOAD DSNHDECP, consider changing the application program to use the DECP address that is returned by ICFID 373, DSNALI, or DSNRLI.

About this task

By using the DECP address that is returned by IFCID 373, DSNALI, or DSNRLI, guarantees that you are using the same DECP module that was used to start Db2. It also allows the code to skip the LOAD entirely, only after successfully connecting to Db2. DSNHDECP is loaded by Db2 into Global, pageable storage, so all programs can share it.

Compatibility of SQL and language data types

The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

When deciding the data types of host variables, consider the following rules and recommendations:

- Numeric data types are compatible with each other:

  Assembler
  A SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT column is compatible with a numeric assembler host variable.

  Fortran
  An INTEGER column is compatible with any Fortran host variable that is defined as INTEGER*2, INTEGER*4, REAL, REAL*4, REAL*8, or DOUBLE PRECISION.

  PL/I
  A SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT column is compatible with a PL/I host variable of BIN FIXED(15), BIN FIXED(31), DECIMAL(s,p), or BIN FLOAT(*n*), where *n* is from 1 to 53, or DEC FLOAT(*m*) where *m* is from 1 to 16.

- Character data types are compatible with each other:

  Assembler
  A CHAR, VARCHAR, or CLOB column is compatible with a fixed-length or varying-length assembler character host variable.

  C/C++
  A CHAR, VARCHAR, or CLOB column is compatible with a single-character, NUL-terminated, or VARCHAR structured form of a C character host variable.

  COBOL
  A CHAR, VARCHAR, or CLOB column is compatible with a fixed-length or varying-length COBOL character host variable.

  Fortran
  A CHAR, VARCHAR, or CLOB column is compatible with Fortran character host variable.

  PL/I
  A CHAR, VARCHAR, or CLOB column is compatible with a fixed-length or varying-length PL/I character host variable.

- Character data types are partially compatible with CLOB locators. You can perform the following assignments:

  – Assign a value in a CLOB locator to a CHAR or VARCHAR column

  – Use a SELECT INTO statement to assign a CHAR or VARCHAR column to a CLOB locator host variable.

  – Assign a CHAR or VARCHAR output parameter from a user-defined function or stored procedure to a CLOB locator host variable.

  – Use a SET assignment statement to assign a CHAR or VARCHAR transition variable to a CLOB locator host variable.

  – Use a VALUES INTO statement to assign a CHAR or VARCHAR function parameter to a CLOB locator host variable.

  However, you cannot use a FETCH statement to assign a value in a CHAR or VARCHAR column to a CLOB locator host variable.

- Graphic data types are compatible with each other:

  Assembler
  A GRAPHIC, VARGRAPHIC, or DBCLOB column is compatible with a fixed-length or varying-length assembler graphic character host variable.

C/C++
A GRAPHIC, VARGRAPHIC, or DBCLOB column is compatible with a single character, NUL-terminated, or VARGRAPHIC structured form of a C graphic host variable.

COBOL
A GRAPHIC, VARGRAPHIC, or DBCLOB column is compatible with a fixed-length or varying-length COBOL graphic string host variable.

PL/I
A GRAPHIC, VARGRAPHIC, or DBCLOB column is compatible with a fixed-length or varying-length PL/I graphic character host variable.

- Graphic data types are partially compatible with DBCLOB locators. You can perform the following assignments:
  - Assign a value in a DBCLOB locator to a GRAPHIC or VARGRAPHIC column
  - Use a SELECT INTO statement to assign a GRAPHIC or VARGRAPHIC column to a DBCLOB locator host variable.
  - Assign a GRAPHIC or VARGRAPHIC output parameter from a user-defined function or stored procedure to a DBCLOB locator host variable.
  - Use a SET assignment statement to assign a GRAPHIC or VARGRAPHIC transition variable to a DBCLOB locator host variable.
  - Use a VALUES INTO statement to assign a GRAPHIC or VARGRAPHIC function parameter to a DBCLOB locator host variable.

  However, you cannot use a FETCH statement to assign a value in a GRAPHIC or VARGRAPHIC column to a DBCLOB locator host variable.
- Binary data types are compatible with each other.
- Binary data types are partially compatible with BLOB locators. You can perform the following assignments:
  - Assign a value in a BLOB locator to a BINARY or VARBINARY column.
  - Use a SELECT INTO statement to assign a BINARY or VARBINARY column to a BLOB locator host variable.
  - Assign a BINARY or VARBINARY output parameter from a user-defined function or stored procedure to a BLOB locator host variable.
  - Use a SET assignment statement to assign a BINARY or VARBINARY transition variable to a BLOB locator host variable.
  - Use a VALUES INTO statement to assign a BINARY or VARBINARY function parameter to a BLOB locator host variable.

  However, you cannot use a FETCH statement to assign a value in a BINARY or VARBINARY column to a BLOB locator host variable.
- Datetime data types are compatible with character host variables.

Fortran
A BINARY, VARBINARY, or BLOB column or BLOB locator is compatible only with a BLOB host variable.

C:
For varying-length BIT data, use BINARY. Some C string manipulation functions process NUL-terminated strings and other functions process strings that are not NUL-terminated. The C string manipulation functions that process NUL-terminated strings cannot handle bit data because these functions might misinterpret a NUL character to be a NUL-terminator.

Assembler
A DATE, TIME, or TIMESTAMP column is compatible with a fixed-length or varying-length assembler character host variable.

C/C++
: A DATE, TIME, or TIMESTAMP column is compatible with a single-character, NUL-terminated, or VARCHAR structured form of a C character host variable.

COBOL
: A DATE, TIME, or TIMESTAMP column is compatible with a fixed-length or varying length COBOL character host variable.

Fortran
: A DATE, TIME, or TIMESTAMP column is compatible with a Fortran character host variable.

PL/I
: A DATE, TIME, or TIMESTAMP column is compatible with a fixed-length or varying-length PL/I character host variable.

-
- The ROWID column is compatible only with a ROWID host variable.
- A host variable is compatible with a distinct type if the host variable type is compatible with the source type of the distinct type.
- XML columns are compatible with the XML host variable types, character types, and binary string types.

  Recommendation: Use the XML host variable types for data from XML columns.
- **Assembler**
  You can assign LOB data to a file reference variable (BLOB_FILE, CLOB_FILE, and DBCLOB_FILE).

When necessary, Db2 automatically converts a fixed-length string to a varying-length string, or a varying-length string to a fixed-length string.

Related concepts

Distinct types
A *distinct type* is a user-defined data type that shares its internal representation with a built-in data type (its *source type*), but is considered to be a separate and incompatible data type for most operations.

Host variable data types for XML data in embedded SQL applications (Db2 Programming for XML)

Related reference

Equivalent SQL and assembler data types
When you declare host variables in your assembler programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, ensure that the host variable is of an equivalent data type.

Equivalent SQL and C data types
When you declare host variables in your C programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

Equivalent SQL and COBOL data types
When you declare host variables in your COBOL programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

Equivalent SQL and Fortran data types
When you declare host variables in your Fortran programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, ensure that the host variable is of an equivalent data type.

Equivalent SQL and PL/I data types
When you declare host variables in your PL/I programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

Equivalent SQL and REXX data types

All REXX data is string data. Therefore, when a REXX program assigns input data to a column, Db2 converts the data from a string type to the column type. When a REXX program assigns column data to an output variable, Db2 converts the data from the column type to a string type.

Using host variables in SQL statements

Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

When you use host variables, adhere to the following requirements:

- You must declare the name of the host variable in the host program before you use it. Host variables follow the naming conventions of the host language.
- You can use a host variable to represent a data value, but you cannot use it to represent a table, view, or column name. You can specify table, view, or column names at run time by using dynamic SQL.
- To use a host variable in an SQL statement, you can specify any valid host variable name that is declared according to the rules of the host language.
- A colon (:) must precede host variables that are used in SQL statements so that Db2 can distinguish a variable name from a column name. When host variables are used outside of SQL statements, do not precede them with a colon. PL/I programs have the following exceptions: If the SQL statement meets any of the following conditions, do not precede a host variable or host variable array in that statement with a colon:
  - The SQL statement is in a program that also contains a DECLARE VARIABLE statement.
  - The host variable is part of a string expression, but the host variable is not the only component of the string expression.
- To optimize performance, make sure that the host language declaration maps as closely as possible to the data type of the associated data in the database.
- For assignments and comparisons between a Db2 column and a host variable of a different data type or length, expect conversions to occur.

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Assignment and comparison (Db2 SQL)

Host variables (Db2 SQL)

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Retrieving a single row of data into host variables

If you know that your query returns only one row, you can specify one or more host variables to contain the column values of the retrieved row.

About this task

Restriction: These instructions do not apply if you do not know how many rows Db2 will return or if you expect Db2 to return more than one row. In these situations, use a cursor. A cursor enables an application to return a set of rows and fetch either one row at a time or one rowset at a time from the result table.

Procedure

In the SELECT statement specify the INTO clause with the name of one or more host variables to contain the retrieved values. Specify one variable for each value that is to be retrieved. The retrieved value can be a column value, a value of a host variable, the result of an expression, or the result of an aggregate function.

Recommendation: If you want to ensure that only one row is returned, specify the FETCH FIRST 1 ROW ONLY clause. Consider using the ORDER BY clause to control which row is returned. If you specify both the ORDER BY clause and the FETCH FIRST clause, ordering is performed on the entire result set before the first row is returned.

Db2 assigns the first value in the result row to the first variable in the list, the second value to the second variable, and so on.

If the SELECT statement returns more than one row, Db2 returns an error, and any data that is returned is undefined and unpredictable.

Examples

Example: Retrieving a single row into a host variable

Suppose that you are retrieving the LASTNAME and WORKDEPT column values from the DSN8B10.EMP table for a particular employee. You can define a host variable in your program to hold each column value and then name the host variables in the INTO clause of the SELECT statement, as shown in the following COBOL example.

```
MOVE '000110' TO CBLEMPNO.
EXEC SQL
  SELECT LASTNAME, WORKDEPT
    INTO :CBLNAME, :CBLDEPT
    FROM DSN8B10.EMP
    WHERE EMPNO = :CBLEMPNO
END-EXEC.
```

In this example, the host variable CBLEMPNO is preceded by a colon (:) in the SQL statement, but it is not preceded by a colon in the COBOL MOVE statement.

This example also uses a host variable to specify a value in a search condition. The host variable CBLEMPNO is defined for the employee number, so that you can retrieve the name and the work department of the employee whose number is the same as the value of the host variable, CBLEMPNO; in this case, 000110.

In the DATA DIVISION section of a COBOL program, you must declare the host variables CBLEMPNO, CBLNAME, and CBLDEPT to be compatible with the data types in the columns EMPNO, LASTNAME, and WORKDEPT of the DSN8B10.EMP table.

Example: Ensuring that a query returns only a single row

You can use the FETCH FIRST 1 ROW ONLY clause in a SELECT statement to ensure that only one row is returned. This action prevents undefined and unpredictable data from being returned when you specify the INTO clause of the SELECT statement. The following example SELECT statement ensures that only one row of the DSN8B10.EMP table is returned.

```
EXEC SQL
  SELECT LASTNAME, WORKDEPT
    INTO :CBLNAME, :CBLDEPT
    FROM DSN8B10.EMP
    FETCH FIRST 1 ROW ONLY
END-EXEC.
```

You can include an ORDER BY clause in the preceding example to control which row is returned. The following example SELECT statement ensures that the only row returned is the one with a last name that is first alphabetically.

```
EXEC SQL
   SELECT LASTNAME, WORKDEPT
     INTO :CBLNAME, :CBLDEPT
     FROM DSN8810.EMP
     ORDER BY LASTNAME
     FETCH FIRST 1 ROW ONLY
END-EXEC.
```

Example: Retrieving the results of host variable values and expressions into host variables

When you specify a list of items in the SELECT clause, that list can include more than the column names of tables and views. You can request a set of column values mixed with host variable values and constants. For example, the following query requests the values of several columns (EMPNO, LASTNAME, and SALARY), the value of a host variable (RAISE), and the value of the sum of a column and a host variable (SALARY and RAISE). For each of these five items in the SELECT list, a host variable is listed in the INTO clause.

```
MOVE 4476 TO RAISE.
MOVE '000220' TO PERSON.
EXEC SQL
  SELECT EMPNO, LASTNAME, SALARY, :RAISE, SALARY + :RAISE
    INTO :EMP-NUM, :PERSON-NAME, :EMP-SAL, :EMP-RAISE, :EMP-TTL
    FROM DSN8B10.EMP
    WHERE EMPNO = :PERSON
END-EXEC.
```

The preceding SELECT statement returns the following results. The column headings represent the names of the host variables.

```
EMP-NUM    PERSON-NAME    EMP-SAL      EMP-RAISE    EMP-TTL
=======    ===========    =======      =========    =======
 000220    LUTZ             29840           4476      34316
```

Example: Retrieving the result of an aggregate function into a host variable

A query can request summary values to be returned from aggregate functions and store those values in host variables. For example, the following query requests that the result of the AVG function be stored in the AVG-SALARY host variable.

```
MOVE 'D11' TO DEPTID.
EXEC SQL
  SELECT WORKDEPT, AVG(SALARY)
    INTO :WORK-DEPT, :AVG-SALARY
    FROM DSN8B10.EMP
    WHERE WORKDEPT = :DEPTID
END-EXEC.
```

Related tasks

Retrieving a set of rows by using a cursor

In an application program, you can retrieve a set of rows from a table or a result table that is returned by a stored procedure. You can retrieve one or more rows at a time.

Related reference

SELECT INTO (Db2 SQL)

Determining whether a retrieved value in a host variable is null or truncated

Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Before you begin

Before you determine whether a retrieved column value is null or truncated, you must have defined the appropriate indicator variables, arrays, and structures.

About this task

An error occurs if you do not use an indicator variable and Db2 retrieves a null value.

Procedure

Determine the value of the indicator variable, array, or structure that is associated with the host variable, array, or structure.

Those values have the following meanings:

Value of indicator variable	Meaning
Less than zero	The column value is null. The value of the host variable does not change from its previous value. If the indicator variable value is -2, the column value is null because of a numeric or character conversion error,
Zero	The column value is nonnull. If the column value is a character string, the retrieved value is not truncated.
Positive integer	The retrieved value is truncated. The integer is the original length of the string.

Table 89. Meanings of values in indicator variables

Examples

Example of testing an indicator variable

Assume that you have defined the following indicator variable INDNULL for the host variable CBLPHONE.

```
EXEC SQL
  SELECT PHONENO
    INTO :CBLPHONE:INDNULL
    FROM DSN8B10.EMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

You can then test INDNULL for a negative value. If the value is negative, the corresponding value of PHONENO is null, and you can disregard the contents of CBLPHONE.

Example of testing an indicator variable array

Suppose that you declare the following indicator array INDNULL for the host-variable array CBLPHONE.

```
EXEC SQL
  FETCH NEXT ROWSET CURS1
    FOR 10 ROWS
    INTO :CBLPHONE :INDNULL
END-EXEC.
```

After the multiple-row FETCH statement, you can test each element of the INDNULL array for a negative value. If an element is negative, you can disregard the contents of the corresponding element in the CBLPHONE host-variable array.

Example of testing an indicator structure in COBOL

The following example defines the indicator structure EMP-IND as an array that contains six values and corresponds to the PEMP-ROW host structure.

```
01 PEMP-ROW.
    10 EMPNO              PIC X(6).
    10 FIRSTNME.
       49 FIRSTNME-LEN    PIC S9(4) USAGE COMP.
       49 FIRSTNME-TEXT   PIC X(12).
    10 MIDINIT            PIC X(1).
    10 LASTNAME.
       49 LASTNAME-LEN    PIC S9(4) USAGE COMP.
       49 LASTNAME-TEXT   PIC X(15).
    10 WORKDEPT           PIC X(3).
    10 EMP-BIRTHDATE      PIC X(10).
01 INDICATOR-TABLE.
    02 EMP-IND            PIC S9(4) COMP OCCURS 6 TIMES.
⋮
MOVE '000230' TO EMPNO.
⋮
EXEC SQL
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, BIRTHDATE
```

```
    INTO :PEMP-ROW:EMP-IND
    FROM DSN8B10.EMP
    WHERE EMPNO = :EMPNO
END-EXEC.
```

You can test the indicator structure EMP-IND for negative values. If, for example, EMP-IND(6) contains a negative value, the corresponding host variable in the host structure (EMP-BIRTHDATE) contains a null value.

Related concepts

Arithmetic and conversion errors
You can track arithmetic and conversion errors by using indicator variables. An indicator variable contains a small integer value that indicates some information about the associated host variable.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Updating data by using host variables

When you want to update a value in a Db2 table, but you do not know the exact value until the program runs, use host variables. Db2 can change a table value to match the current value of the host variable.

Procedure

To update data by using host variables:

1. Declare the necessary host variables.
2. Specify an UPDATE statement with the appropriate host variable names in the SET clause.

Examples

Example of updating a single row by using a host variable

The following COBOL example changes an employee's phone number to the value in the NEWPHONE host variable. The employee ID value is passed through the EMPID host variable.

```
MOVE '4246' TO NEWPHONE.
MOVE '000110' TO EMPID.
EXEC SQL
  UPDATE DSN8B10.EMP
    SET PHONENO = :NEWPHONE
    WHERE EMPNO = :EMPID
END-EXEC.
```

Example of updating a single row by using a host variable

The following example gives the employees in a particular department a salary increase of 10%. The department value is passed through the DEPTID host variable.

```
MOVE 'D11' TO DEPTID.
EXEC SQL
  UPDATE DSN8B10.EMP
    SET SALARY = 1.10 * SALARY
    WHERE WORKDEPT = :DEPTID
END-EXEC.
```

Related reference

UPDATE (Db2 SQL)

Inserting a single row by using a host variable

Use host variables in your INSERT statement when you don't know at least some of the values to insert until the program runs.

About this task

Restriction: These instructions apply only to inserting a single row. If you want to insert multiple rows, use host variable arrays or the form of the INSERT statement that selects values from another table or view.

Procedure

Specify an INSERT statement with column values in the VALUES clause. Specify host variables or a combination of host variables and constants as the column values.

Db2 inserts the first value into the first column in the list, the second value into the second column, and so on.

Example

The following example uses host variables to insert a single row into the activity table.

```
EXEC SQL
  INSERT INTO DSN8B10.ACT
    VALUES (:HV-ACTNO, :HV-ACTKWD, :HV-ACTDESC)
END-EXEC.
```

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Related reference

INSERT (Db2 SQL)

Using host-variable arrays in SQL statements

Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

To use a host-variable array in an SQL statement, specify any valid host-variable array that is declared according to the host language rules. You can specify host-variable arrays in C or C++, COBOL, and PL/I. You must declare the array in the host program before you use it.

Host-variable arrays are defined by statements of the host language, as explained in the following topics:

- "Host-variable arrays in C and C++" on page 595
- "Host-variable arrays in COBOL" on page 660
- "Host-variable arrays in PL/I" on page 712

Tip: Host-variable arrays are not supported for assembler, FORTRAN, or REXX programs. However, you can use SQL descriptor areas (SQLDA) to achieve similar results in any host language. For more information see "Defining SQL descriptor areas (SQLDA)" on page 476.

Host-variable arrays can be referenced only as a simple reference in the following contexts. In syntax diagrams, *host-variable-array* designates a reference to a host-variable array.

- In a FETCH statement for a multiple-row fetch. See FETCH (Db2 SQL).
- In an INSERT statement with multiple rows of source data. This is also referred to as multiple-row insert. See INSERT (Db2 SQL).

- In a MERGE statement with multiple rows of source data. See MERGE (Db2 SQL).
- In an EXECUTE statement to provide a value for a parameter marker in a dynamic multi-row INSERT or MERGE. See EXECUTE (Db2 SQL).

Related concepts

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Host-variable arrays
You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Retrieving multiple rows of data into host-variable arrays

If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

About this task

You can use host-variable arrays to specify a program data area to contain multiple rows of column values. A Db2 *rowset cursor* enables an application to retrieve and process a set of rows from the result table of the cursor.

Related concepts

Host-variable arrays
You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Related tasks

Accessing data by using a rowset-positioned cursor
A rowset-positioned cursor is a cursor that can return one or more rows for a single fetch operation. The cursor is positioned on the set of rows that are to be fetched.

Executing SQL statements by using a rowset cursor
You can use rowset cursors to execute multiple-row FETCH statements, positioned UPDATE statements, and positioned DELETE statements.

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Inserting multiple rows of data from host-variable arrays

Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

About this task

You can use a form of the INSERT statement or MERGE statement to insert multiple rows from values that are provided in host-variable arrays. Each array contains values for a column of the target table. The first value in an array corresponds to the value for that column for the first inserted row, the second value in the array corresponds to the value for the column in the second inserted row, and so on. Db2 determines the attributes of the values based on the declaration of the array.

Example

You can insert the number of rows that are specified in the host variable NUM-ROWS by using the following INSERT statement:

```
EXEC SQL
  INSERT INTO DSN8B10.ACT
    (ACTNO, ACTKWD, ACTDESC)
    VALUES (:HVA1, :HVA2, :HVA3)
    FOR :NUM-ROWS ROWS
END-EXEC.
```

Assume that the host-variable arrays HVA1, HVA2, and HVA3 have been declared and populated with the values that are to be inserted into the ACTNO, ACTKWD, and ACTDESC columns. The NUM-ROWS host variable specifies the number of rows that are to be inserted, which must be less than or equal to the dimension of each host-variable array.

Related concepts

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Related tasks

Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Related reference

INSERT (Db2 SQL)

MERGE (Db2 SQL)

Inserting null values into columns by using indicator variables or arrays

If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Procedure

To insert null values into columns by using indicator variables or arrays:

1. Define an indicator variable or array for a particular host variable or array.
2. Assign a negative value to the indicator variable or array.
3. Issue the appropriate INSERT, UPDATE, or MERGE statement with the host variable or array and its indicator variable or array.

   When Db2 processes INSERT, UPDATE, and MERGE statements, it checks the indicator variable if one exists. If the indicator variable is negative, the column value is null. If the indicator variable is greater than -1, the associated host variable contains a value for the column.

Examples

Example of setting a column value to null by using an indicator variable

Suppose your program reads an employee ID and a new phone number and must update the employee table with the new number. The new number could be missing if the old number is incorrect, but a new number is not yet available. If the new value for column PHONENO might be null, you can use an indicator variable, as shown in the following UPDATE statement.

```
EXEC SQL
  UPDATE DSN8B10.EMP
    SET PHONENO = :NEWPHONE:PHONEIND
    WHERE EMPNO = :EMPID
END-EXEC.
```

When NEWPHONE contains a non-null value, set the indicator variable PHONEIND to zero by preceding the UPDATE statement with the following line:

```
MOVE 0 TO PHONEIND.
```

When NEWPHONE contains a null value, set PHONEIND to a negative value by preceding the UPDATE statement with the following line:

```
MOVE -1 TO PHONEIND.
```

Example of setting a column value to null by using an indicator variable array

Assume that host-variable arrays hva1 and hva2 have been populated with values that are to be inserted into the ACTNO and ACTKWD columns. Assume the ACTDESC column allows nulls. To set the ACTDESC column to null, assign -1 to the elements in its indicator array, ind3, as shown in the following example:

```
/* Initialize each indicator array */
for (i=0; i<10; i++) {
  ind1[i] = 0;
  ind2[i] = 0;
  ind3[i] = -1;
}

EXEC SQL
  INSERT INTO DSN8B10.ACT
    (ACTNO, ACTKWD, ACTDESC)
     VALUES (:hva1:ind1, :hva2:ind2, :hva3:ind3)
     FOR 10 ROWS;
```

Db2 ignores the values in the hva3 array and assigns the values in the ARTDESC column to null for the 10 rows that are inserted.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Retrieving a single row of data into a host structure

If you know that your query returns multiple column values for only one row, you can specify a host structure to contain the column values.

About this task

In the following example, assume that your COBOL program includes the following SQL statement:

```
EXEC SQL
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
    INTO :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME, :WORKDEPT
    FROM DSN8B10.VEMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

If you want to avoid listing host variables, you can substitute the name of a structure, say :PEMP, that contains :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME, and :WORKDEPT. The example then reads:

```
EXEC SQL
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
    INTO :PEMP
    FROM DSN8B10.VEMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

You can declare a host structure yourself, or you can use DCLGEN to generate a COBOL record description, PL/I structure declaration, or C structure declaration that corresponds to the columns of a table.

Related concepts

DCLGEN (declarations generator)
Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Host structures
Use host structures to pass a group of host variables between Db2 and your application.

Example: Adding DCLGEN declarations to a library
You can use DCLGEN to generate table and variable declarations for C, COBOL, and PL/I programs. If you store these declarations in a library, you can later integrate them into your program with a single SQL INCLUDE statement.

Including dynamic SQL in your program

Dynamic SQL is prepared and executed while the program is running.

Before you begin

Before you use dynamic SQL, consider whether static SQL or dynamic SQL is the best technique for your application, and consider the type of dynamic SQL that you want to use. Also consider the performance implications of using dynamic SQL in application programs. For information about methods that you can use to improve the performance of dynamic SQL statements, see Improving dynamic SQL performance (Db2 Performance).

About this task

Introductory concepts

Submitting SQL statements to Db2 (Introduction to Db2 for z/OS)
Dynamic SQL applications (Introduction to Db2 for z/OS)

Dynamic SQL prepares and executes the SQL statements within a program, while the program is running.

You can issue dynamic SQL statements in the following contexts:

Interactive SQL
A user enters SQL statements through SPUFI, the command line processor, or an interactive tool, such as QMF for Workstation.Db2 prepares and executes those statements as dynamic SQL statements.

Embedded dynamic SQL
Your application puts the SQL source in host variables and includes PREPARE and EXECUTE statements that tell Db2 to prepare and run the contents of those host variables at run time. You must precompile and bind programs that include embedded dynamic SQL.

Deferred embedded SQL
Deferred embedded SQL statements are neither fully static nor fully dynamic. Like static statements, deferred embedded SQL statements are embedded within applications; however, like dynamic statements, they are prepared at run time. Db2 processes the deferred embedded SQL statements with bind-time rules. For example, Db2 uses the authorization ID and qualifier (that are determined at bind time) as the plan or package owner.

Dynamic SQL executed through ODBC or JDBC functions
Your application contains ODBC function calls that pass dynamic SQL statements as arguments. You do not need to precompile and bind programs that use ODBC function calls.

JDBC application support lets you write dynamic SQL applications in Java.

For most Db2 users, *static SQL*, which is embedded in a host language program and bound before the program runs, provides a straightforward, efficient path to Db2 data. You can use static SQL when you know before run time what SQL statements your application needs to execute.

Related tasks

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Improving dynamic SQL performance by enabling the dynamic statement cache (Db2 Performance)

Related information

Dynamic Statement Cache (white paper)

Differences between static and dynamic SQL

Static and dynamic SQL are each appropriate for different circumstances. You should consider the differences between the two when determining whether static SQL or dynamic SQL is best for your application.

Flexibility of static SQL with host variables

Introductory concepts

Static SQL (Introduction to Db2 for z/OS)
Static SQL applications (Introduction to Db2 for z/OS)
Submitting SQL statements to Db2 (Introduction to Db2 for z/OS)
Dynamic SQL applications (Introduction to Db2 for z/OS)

When you use static SQL, you cannot change the form of SQL statements unless you make changes to the program. However, you can increase the flexibility of static statements by using host variables.

Example: In the following example, the UPDATE statement can update the salary of any employee. At bind time, you know that salaries must be updated, but you do not know until run time whose salaries should be updated, and by how much.

```
01  IOAREA.
    02  EMPID              PIC X(06).
    02  NEW-SALARY         PIC S9(7)V9(2) COMP-3.
⋮ (Other declarations)
READ CARDIN RECORD INTO IOAREA
  AT END MOVE 'N' TO INPUT-SWITCH.
⋮ (Other COBOL statements)
EXEC SQL
  UPDATE DSN8B10.EMP
    SET SALARY = :NEW-SALARY
    WHERE EMPNO = :EMPID
END-EXEC.
```

The statement (UPDATE) does not change, nor does its basic structure, but the input can change the results of the UPDATE statement.

Flexibility of dynamic SQL

What if a program must use different types and structures of SQL statements? If there are so many types and structures that it cannot contain a model of each one, your program might need dynamic SQL.

You can use one of the following programs to execute dynamic SQL:

Db2 Query Management Facility (QMF)
Provides an alternative interface to Db2 that accepts almost any SQL statement

SPUFI
Accepts SQL statements from an input data set, and then processes and executes them dynamically

command line processor
Accepts SQL statements from a UNIX System Services environment.

Limitations of dynamic SQL

You cannot use some of the SQL statements dynamically.

Dynamic SQL processing

A program that provides for dynamic SQL accepts as input, or generates, an SQL statement in the form of a character string. You can simplify the programming if you can plan the program not to use SELECT statements, or to use only those that return a known number of values of known types. In the most general case, in which you do not know in advance about the SQL statements that will execute, the program typically takes these steps:

1. Translates the input data, including any parameter markers, into an SQL statement
2. Prepares the SQL statement to execute and acquires a description of the result table
3. Obtains, for SELECT statements, enough main storage to contain retrieved data
4. Executes the statement or fetches the rows of data
5. Processes the information returned
6. Handles SQL return codes.

Performance of static and dynamic SQL

To access Db2 data, an SQL statement requires an access path. Two big factors in the performance of an SQL statement are the amount of time that Db2 uses to determine the access path at run time and whether the access path is efficient. Db2 determines the access path for a statement at either of these times:

- When you bind the plan or package that contains the SQL statement
- When the SQL statement executes

The time at which Db2 determines the access path depends on these factors:

- Whether the statement is executed statically or dynamically
- Whether the statement contains input host variables
- Whether the statement contains a declared global temporary table.

Static SQL statements with no input host variables

For static SQL statements that do not contain input host variables, Db2 determines the access path when you bind the plan or package. This combination yields the best performance because the access path is already determined when the program executes.

Static SQL statements with input host variables

For static SQL statements that have input host variables, the time at which Db2 determines the access path depends on the REOPT bind option that you specify: REOPT(NONE) or REOPT(ALWAYS). REOPT(NONE) is the default. Do not specify REOPT(AUTO) or REOPT(ONCE); these options are applicable only to dynamic statements. Db2 ignores REOPT(ONCE) and REOPT(AUTO) for static SQL statements, because Db2 caches only dynamic SQL statements.

If you specify REOPT(NONE), Db2 determines the access path at bind time, just as it does when there are no input variables.

If you specify REOPT(ALWAYS), Db2 determines the access path at bind time and again at run time, using the values of the following types of input variables:

- Host variables
- Parameter markers
- Special registers

Db2 must spend extra time determining the access path for statements at run time. However if Db2 determines a significantly better access path using the variable values, you might see an overall performance improvement. With REOPT(ALWAYS), Db2 optimizes statements using known literal values. Knowing the literal values can help Db2 to choose a more efficient access path when the columns contain skewed data. Db2 can also recognize which partitions qualify if there are search conditions with host variables on the limit keys of partitioned table spaces.

With REOPT(ALWAYS) Db2 does not start the optimization over from the beginning. For example Db2 does not perform query transformations based on the literal values. Consequently, static SQL statements that use host variables optimized with REOPT(ALWAYS) and similar SQL statements that use explicit literal values might result in different access paths.

Dynamic SQL statements

For dynamic SQL statements, Db2 determines the access path at run time, when the statement is prepared. The repeating cost of preparing a dynamic statement can make the performance worse than that of static SQL statements. However, if you execute the same SQL statement often, you can use the dynamic statement cache to decrease the number of times that those dynamic statements must be prepared.

Dynamic SQL statements with input host variables

When you bind applications that contain dynamic SQL statements with input host variables, consider using the REOPT(ALWAYS), REOPT(ONCE), or REOPT(AUTO) bind options, instead of the REOPT(NONE) option.

Use REOPT(ALWAYS) when you are not using the dynamic statement cache. Db2 determines the access path for statements at each EXECUTE or OPEN of the statement. This option ensures the best access path for a statement, but using REOPT(ALWAYS) can increase the cost of frequently used dynamic SQL statements.

Consequently, the REOPT(ALWAYS) option is not a good choice for high-volume sub-second queries. For high-volume fast running queries, the repeating cost of prepare can exceed the execution cost of the statement. Statements that are processed under the REOPT(ALWAYS) option are excluded from the dynamic statement cache even if dynamic statement caching is enabled because Db2 cannot reuse access paths when REOPT(ALWAYS) is specified.

Use REOPT(ONCE) or REOPT(AUTO) when you are using the dynamic statements cache:

- If you specify REOPT(ONCE), Db2 determines and the access path for statements only at the first EXECUTE or OPEN of the statement. It saves that access path in the dynamic statement cache and uses it until the statement is invalidated or removed from the cache. This reuse of the access path reduces the prepare cost of frequently used dynamic SQL statements that contain input host variables; however, it does not account for changes to parameter marker values for dynamic statements.

  The REOPT(ONCE) option is ideal for ad-hoc query applications such as SPUFI, DSNTEP2, DSNTEP4, DSNTIAUL, and QMF Db2 can better optimize statements knowing the literal values for special registers such as CURRENT DATE and CURRENT TIMESTAMP, rather than using default filter factor estimates.
- If you specify REOPT(AUTO), Db2 determines the access path at run time. For each execution of a statement with parameter markers, Db2 generates a new access path if it determines that a new access path is likely to improve performance.

Coding PREPARE statements for efficient optimization

You should code your PREPARE statements to minimize overhead. With REOPT(AUTO), REOPT(ALWAYS), and REOPT(ONCE), Db2 prepares an SQL statement at the same time as it processes OPEN or EXECUTE for the statement. That is, Db2 processes the statement as if you specify DEFER(PREPARE). However, Db2 prepares the statement twice in the following situations:

- Your program issues the DESCRIBE statement before the OPEN statement
- You issue the PREPARE statement with the INTO parameter

For the first prepare, Db2 determines the access path without using input variable values. For the second prepare, Db2 uses the input variable values to determine the access path. This extra prepare can decrease performance.

If you specify REOPT(ALWAYS), Db2 prepares the statement twice each time it is run.

If you specify REOPT(ONCE), Db2 prepares the statement twice only when the statement has never been saved in the cache. If the statement has been prepared and saved in the cache, Db2 will use the saved version of the statement to complete the DESCRIBE statement.

If you specify REOPT(AUTO), Db2 initially prepares the statement without using input variable values. If the statement has been saved in the cache, for the subsequent OPEN or EXECUTE, Db2 determines if a new access path is needed according to the input variable values.

For a statement that uses a cursor, you can avoid the double prepare by placing the DESCRIBE statement after the OPEN statement in your program.

If you use predictive governing, and a dynamic SQL statement that is bound with either REOPT(ALWAYS) or REOPT(ONCE) exceeds a predictive governing warning threshold, your application does not receive a warning SQLCODE. However, it will receive an error SQLCODE from the OPEN or EXECUTE statement.

Related tasks

Reoptimizing SQL statements at run time (Db2 Performance)

Improving dynamic SQL performance by enabling the dynamic statement cache (Db2 Performance)

Related reference

Characteristics of SQL statements in Db2 for z/OS (Db2 SQL)

REOPT bind option (Db2 Commands)

Possible host languages for dynamic SQL applications

Programs that use dynamic SQL are usually written in assembler, C, PL/I, REXX, and COBOL. All SQL statements in REXX programs are considered dynamic SQL.

You can write non-SELECT and fixed-list SELECT statements in any of the Db2 supported languages. A program containing a varying-list SELECT statement is more difficult to write in Fortran, because the program cannot run without the help of a subroutine to manage address variables (pointers) and storage allocation.

Most of the examples in this topic are in PL/I. Longer examples in the form of complete programs are available in the sample applications:

DSNTEP2
Processes both SELECT and non-SELECT statements dynamically. (PL/I).

DSNTIAD
Processes only non-SELECT statements dynamically. (Assembler).

DSNTIAUL
Processes SELECT statements dynamically. (Assembler).

Library *prefix*.SDSNSAMP contains the sample programs. You can view the programs online, or you can print them using ISPF, IEBPTPCH, or your own printing program.

You can use all forms of dynamic SQL in all supported versions of COBOL.

Related concepts

Sample COBOL dynamic SQL program

You can code dynamic varying-list SELECT statements in a COBOL program. *Varying-List SELECT statements* are statements for which you do not know the number or data types of columns that are to be returned when you write the program.

Including dynamic SQL for non-SELECT statements in your program

The easiest way to use dynamic SQL is to use non-SELECT statements. Because you do not need to dynamically allocate any main storage, you can write your program in any host language, including Fortran.

Procedure

Your program must take the following steps:

1. Include an SQLCA. The requirements for an SQL communications area (SQLCA) are the same as for static SQL statements. For REXX, Db2 includes the SQLCA automatically.
2. Load the input SQL statement into a data area. The procedure for building or reading the input SQL statement is not discussed here; the statement depends on your environment and sources of information. You can read in complete SQL statements, or you can get information to build the statement from data sets, a user at a terminal, previously set program variables, or tables in the database.

   If you attempt to execute an SQL statement dynamically that Db2 does not allow, you get an SQL error.
3. Execute the statement. You can use either of these methods:
   - EXECUTE IMMEDIATE
   - PREPARE and EXECUTE
4. Handle any errors that might result. The requirements are the same as those for static SQL statements. The return code from the most recently executed SQL statement appears in the host variables SQLCODE and SQLSTATE or corresponding fields of the SQLCA.

Related concepts

Sample dynamic and static SQL in a C program
Programs that access Db2 can contain static SQL, dynamic SQL, or both.

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements
You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Dynamically executing an SQL statement by using EXECUTE IMMEDIATE

In certain situations, you might want your program to prepare and dynamically execute a statement immediately after reading it.

Dynamically executing an SQL statement by using PREPARE and EXECUTE
As an alternative to executing an SQL statement immediately after it is read, you can prepare and execute the SQL statement in two steps. This two-step method is useful when you need to execute an SQL statement multiple times with different values.

Including dynamic SQL for fixed-list SELECT statements in your program

A fixed-list SELECT statement returns rows that contain a known number of values of a known type. When you use this type of statement, you know in advance exactly what kinds of host variables you need to declare to store the results.

About this task

The term "fixed-list" does not imply that you must know in advance how many rows of data will be returned. However, you must know the number of columns and the data types of those columns. A fixed-list SELECT statement returns a result table that can contain any number of rows; your program looks at those rows one at a time, using the FETCH statement. Each successive fetch returns the same number of values as the last, and the values have the same data types each time. Therefore, you can specify host variables as you do for static SQL.

An advantage of the fixed-list SELECT is that you can write it in any of the programming languages that Db2 supports. Varying-list dynamic SELECT statements require assembler, C, PL/I, and COBOL.

For example, suppose that your program retrieves last names and phone numbers by dynamically executing SELECT statements of this form:

```
SELECT LASTNAME, PHONENO FROM DSN8B10.EMP
  WHERE ... ;
```

The program reads the statements from a terminal, and the user determines the WHERE clause.

As with non-SELECT statements, your program puts the statements into a varying-length character variable; call it DSTRING. Eventually you prepare a statement from DSTRING, but first you must declare a cursor for the statement and give it a name.

Procedure

To execute a fixed-list SELECT statement dynamically, your program must:

1. Include an SQLCA.
2. Load the input SQL statement into a data area.

   The preceding two steps are exactly the same including dynamic SQL for non-SELECT statements in your program.
3. Declare a cursor for the statement name.

   Dynamic SELECT statements cannot use INTO. Therefore, you must use a cursor to put the results into host variables.

   For example, when you declare the cursor, use the statement name (call it STMT), and give the cursor itself a name (for example, C1):

   ```
   EXEC SQL DECLARE C1 CURSOR FOR STMT;
   ```
4. Prepare the statement.
   Prepare a statement (STMT) from DSTRING. This is one possible PREPARE statement:

   ```
   EXEC SQL PREPARE STMT FROM :DSTRING ATTRIBUTES :ATTRVAR;
   ```

   ATTRVAR contains attributes that you want to add to the SELECT statement, such as FETCH FIRST 10 ROWS ONLY or OPTIMIZE for 1 ROW. In general, if the SELECT statement has attributes that conflict with the attributes in the PREPARE statement, the attributes on the SELECT statement take precedence over the attributes on the PREPARE statement. However, in this example, the SELECT statement in DSTRING has no attributes specified, so Db2 uses the attributes in ATTRVAR for the SELECT statement.

As with non-SELECT statements, the fixed-list SELECT could contain parameter markers. However, this example does not need them.

5. Open the cursor.

   The OPEN statement evaluates the SELECT statement named STMT.

   For example, without parameter markers, use this statement:

   ```
   EXEC SQL OPEN C1;
   ```

   If STMT contains parameter markers, you must use the USING clause of OPEN to provide values for all of the parameter markers in STMT. If four parameter markers are in STMT, you need the following statement:

   ```
   EXEC SQL OPEN C1 USING :PARM1, :PARM2, :PARM3, :PARM4;
   ```

6. Fetch rows from the result table.
   For example, your program could repeatedly execute a statement such as this:

   ```
   EXEC SQL FETCH C1 INTO :NAME, :PHONE;
   ```

   The key feature of this statement is the use of a list of host variables to receive the values returned by FETCH. The list has a known number of items (in this case, two items, :NAME and :PHONE) of known data types (both are character strings, of lengths 15 and 4, respectively).

   You can use this list in the FETCH statement only because you planned the program to use only fixed-list SELECTs. Every row that cursor C1 points to must contain exactly two character values of appropriate length. If the program is to handle anything else, it must use the techniques for including dynamic SQL for varying-list SELECT statements in your program.

7. Close the cursor.

   This step is the same as for static SQL.

   A WHENEVER NOT FOUND statement in your program can name a routine that contains this statement:

   ```
   EXEC SQL CLOSE C1;
   ```

8. Handle any resulting errors. This step is the same as for static SQL, except for the number and types of errors that can result.

Related concepts

Sample dynamic and static SQL in a C program
Programs that access Db2 can contain static SQL, dynamic SQL, or both.

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements

You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related tasks

Including dynamic SQL for non-SELECT statements in your program
The easiest way to use dynamic SQL is to use non-SELECT statements. Because you do not need to dynamically allocate any main storage, you can write your program in any host language, including Fortran.

Including dynamic SQL for varying-list SELECT statements in your program
A varying-list SELECT statement returns rows that contain an unknown number of values of unknown type. When you use this type of statement, you do not know in advance exactly what kinds of host variables you need to declare for storing the results.

Including dynamic SQL for varying-list SELECT statements in your program

A varying-list SELECT statement returns rows that contain an unknown number of values of unknown type. When you use this type of statement, you do not know in advance exactly what kinds of host variables you need to declare for storing the results.

About this task

Because the varying-list SELECT statement requires pointer variables for the SQL descriptor area, you cannot issue it from a Fortran program. A Fortran program can call a subroutine written in a language that supports pointer variables (such as PL/I or assembler), if you need to use a varying-list SELECT statement.

Procedure

To execute a varying-list SELECT statement dynamically, your program must follow these steps:

1. Include an SQLCA.

   Db2 performs this step for a REXX program.
2. Load the input SQL statement into a data area.
3. Prepare and execute the statement. This step is more complex than for fixed-list SELECTs. It involves the following steps:
   a) Include an SQLDA (SQL descriptor area).

      Db2 performs this step for a REXX program.
   b) Declare a cursor and prepare the variable statement.
   c) Obtain information about the data type of each column of the result table.
   d) Determine the main storage needed to hold a row of retrieved data.
      You do not perform this step for a REXX program.
   e) Put storage addresses in the SQLDA to tell where to put each item of retrieved data.
   f) Open the cursor.
   g) Fetch a row.
   h) Eventually close the cursor and free main storage.

   Additional complications exist for statements with parameter markers.
4. Handle any errors that might result.

Examples

Preparing a varying-list SELECT statement

Suppose that your program dynamically executes SQL statements, but this time without any limits on their form. Your program reads the statements from a terminal, and you know nothing about them in advance. They might not even be SELECT statements.

As with non-SELECT statements, your program puts the statements into a varying-length character variable; call it DSTRING. Your program goes on to prepare a statement from the variable and then give the statement a name; call it STMT.

Now, the program must find out whether the statement is a SELECT. If it is, the program must also find out how many values are in each row, and what their data types are. The information comes from an SQL descriptor area (SQLDA).

SQL descriptor area (SQLDA)

The SQLDA is a structure that is used to communicate with your program, and storage for it is usually allocated dynamically at run time.

To include the SQLDA in a PL/I or C program, use:

```
EXEC SQL INCLUDE SQLDA;
```

For assembler, use this in the storage definition area of a CSECT:

```
EXEC SQL INCLUDE SQLDA
```

For COBOL, use:

```
EXEC SQL INCLUDE SQLDA END-EXEC.
```

You cannot include an SQLDA in a Fortran, or REXX program.

Obtaining information about the SQL statement

An SQLDA can contain a variable number of occurrences of SQLVAR, each of which is a set of five fields that describe one column in the result table of a SELECT statement.

The number of occurrences of SQLVAR depends on the following factors:

- The number of columns in the result table you want to describe.
- Whether you want the PREPARE or DESCRIBE to put both column names and labels in your SQLDA. This is the option USING BOTH in the PREPARE or DESCRIBE statement.
- Whether any columns in the result table are LOB types or distinct types.

The following table shows the minimum number of SQLVAR instances you need for a result table that contains *n* columns.

Table 90. Minimum number of SQLVARs for a result table with n columns

Type of DESCRIBE and contents of result table	Not USING BOTH	USING BOTH
No distinct types or LOBs	*n*	2**n*
Distinct types but no LOBs	2**n*	3**n*
LOBs but no distinct types	2**n*	2**n*
LOBs and distinct types	2**n*	3**n*

An SQLDA with *n* occurrences of SQLVAR is referred to as a *single SQLDA*, an SQLDA with 2**n* occurrences of SQLVAR a *double SQLDA*, an SQLDA with 3**n* occurrences of SQLVAR a *triple SQLDA*.

A program that admits SQL statements of every kind for dynamic execution has two choices:

- Provide the largest SQLDA that it could ever need. The maximum number of columns in a result table is 750, so an SQLDA for 750 columns occupies 33 016 bytes for a single SQLDA, 66 016 bytes for a double SQLDA, or 99 016 bytes for a triple SQLDA. Most SELECT statements do not retrieve 750 columns, so the program does not usually use most of that space.
- Provide a smaller SQLDA, with fewer occurrences of SQLVAR. From this the program can find out whether the statement was a SELECT and, if it was, how many columns are in its result table. If more columns are in the result than the SQLDA can hold, Db2 returns no descriptions. When this happens, the program must acquire storage for a second SQLDA that is long enough to hold the column descriptions, and ask Db2 for the descriptions again. Although this technique is more complicated to program than the first, it is more general.

  How many columns should you allow? You must choose a number that is large enough for most of your SELECT statements, but not too wasteful of space; 40 is a good compromise. To illustrate what you must do for statements that return more columns than allowed, the example in this discussion uses an SQLDA that is allocated for at least 100 columns.

Declaring a cursor for the statement

As before, you need a cursor for the dynamic SELECT. For example, write:

```
EXEC SQL
  DECLARE C1 CURSOR FOR STMT;
```

Preparing the statement using the minimum SQLDA

Suppose that your program declares an SQLDA structure with the name MINSQLDA, having 100 occurrences of SQLVAR and SQLN set to 100. To prepare a statement from the character string in DSTRING and also enter its description into MINSQLDA, write this:

```
EXEC SQL PREPARE STMT FROM :DSTRING;
EXEC SQL DESCRIBE STMT INTO :MINSQLDA;
```

Equivalently, you can use the INTO clause in the PREPARE statement:

```
EXEC SQL
  PREPARE STMT INTO :MINSQLDA FROM :DSTRING;
```

Do not use the USING clause in either of these examples. At the moment, only the minimum SQLDA is in use. The following figure shows the contents of the minimum SQLDA in use.

Figure 30. The minimum SQLDA structure

SQLN determines what SQLVAR gets

The SQLN field, which you must set before using DESCRIBE (or PREPARE INTO), tells how many occurrences of SQLVAR the SQLDA is allocated for. If DESCRIBE needs more than that, the results of the DESCRIBE depend on the contents of the result table. Let *n* indicate the number of columns in the result table. Then:

- If the result table contains at least one distinct type column but no LOB columns, you do not specify USING BOTH, and n<=SQLN<2*n, then Db2 returns base SQLVAR information in the first *n* SQLVAR occurrences, but no distinct type information. Base SQLVAR information includes:
  - Data type code
  - Length attribute (except for LOBs)
  - Column name or label
  - Host variable address
  - Indicator variable address
- Otherwise, if SQLN is less than the minimum number of SQLVARs specified in the table above, then Db2 returns no information in the SQLVARs.

Regardless of whether your SQLDA is big enough, whenever you execute DESCRIBE, Db2 returns the following values, which you can use to build an SQLDA of the correct size:

- SQLD is 0 if the SQL statement is not a SELECT. Otherwise, SQLD is the number of columns in the result table. The number of SQLVAR occurrences you need for the SELECT depends on the value in the seventh byte of SQLDAID.
- The seventh byte of SQLDAID is 2 if each column in the result table requires two SQLVAR entries. The seventh byte of SQLDAID is 3 if each column in the result table requires three SQLVAR entries.

If the statement is not a SELECT

To find out if the statement is a SELECT, your program can query the SQLD field in MINSQLDA. If the field contains 0, the statement is not a SELECT, the statement is already prepared, and your program can execute it. If no parameter markers are in the statement, you can use:

```
EXEC SQL EXECUTE STMT;
```

(If the statement does contain parameter markers, you must use an SQL descriptor area)

Acquiring storage for a second SQLDA if needed

Now you can allocate storage for a second, full-size SQLDA; call it FULSQLDA. The following figure shows its structure.

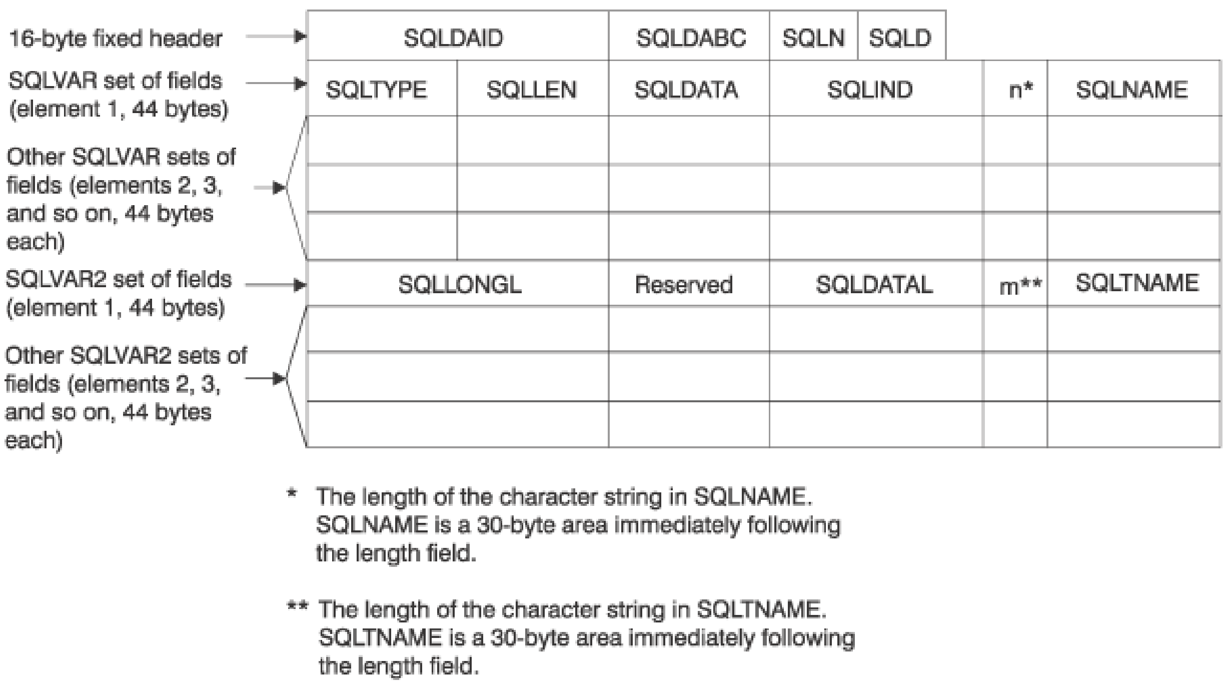


Figure 31. The full-size SQLDA structure

FULSQLDA has a fixed-length header of 16 bytes in length, followed by a varying-length section that consists of structures with the SQLVAR format. If the result table contains LOB columns or distinct type columns, a varying-length section that consists of structures with the SQLVAR2 format follows the structures with SQLVAR format. All SQLVAR structures and SQLVAR2 structures are 44 bytes long. The number of SQLVAR and SQLVAR2 elements you need is in the SQLD field of MINSQLDA, and the total length you need for FULSQLDA (16 + SQLD * 44) is in the SQLDABC field of MINSQLDA. Allocate that amount of storage.

Describing the SELECT statement again

After allocating sufficient space for FULSQLDA, your program must take these steps:

1. Put the total number of SQLVAR and SQLVAR2 occurrences in FULSQLDA into the SQLN field of FULSQLDA. This number appears in the SQLD field of MINSQLDA.
2. Describe the statement again into the new SQLDA:

```
EXEC SQL DESCRIBE STMT INTO :FULSQLDA;
```

After the DESCRIBE statement executes, each occurrence of SQLVAR in the full-size SQLDA (FULSQLDA in our example) contains a description of one column of the result table in five fields. If an SQLVAR occurrence describes a LOB column or distinct type column, the corresponding SQLVAR2 occurrence contains additional information specific to the LOB or distinct type.

The following figure shows an SQLDA that describes two columns that are not LOB columns or distinct type columns.

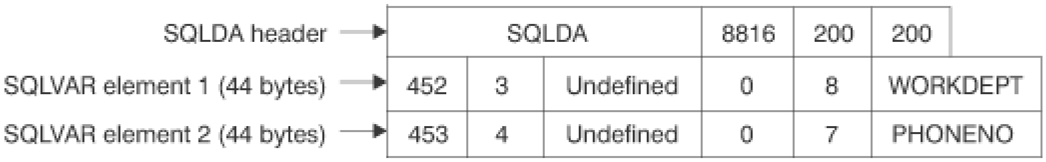


Figure 32. Contents of FULSQLDA after executing DESCRIBE

Acquiring storage to hold a row

Before fetching rows of the result table, your program must:

1. Analyze each SQLVAR description to determine how much space you need for the column value.
2. Derive the address of some storage area of the required size.
3. Put this address in the SQLDATA field.

If the SQLTYPE field indicates that the value can be null, the program must also put the address of an indicator variable in the SQLIND field. The following figures show the SQL descriptor area after you take certain actions.

In the previous figure, the DESCRIBE statement inserted all the values except the first occurrence of the number 200. The program inserted the number 200 before it executed DESCRIBE to tell how many occurrences of SQLVAR to allow. If the result table of the SELECT has more columns than this, the SQLVAR fields describe nothing.

The first SQLVAR pertains to the first column of the result table (the WORKDEPT column). SQLVAR element 1 contains fixed-length character strings and does not allow null values (SQLTYPE=452); the length attribute is 3.

The following figure shows the SQLDA after your program acquires storage for the column values and their indicators, and puts the addresses in the SQLDATA fields of the SQLDA.

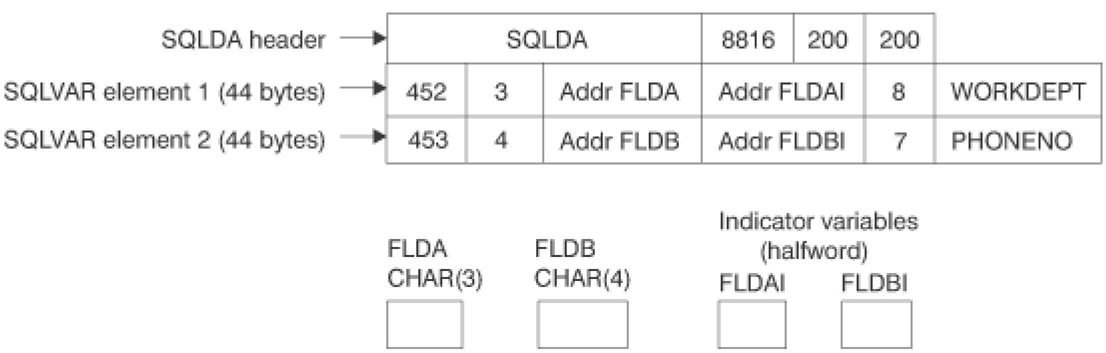


Figure 33. SQL descriptor area after analyzing descriptions and acquiring storage

The following figure shows the SQLDA after your program executes a FETCH statement.

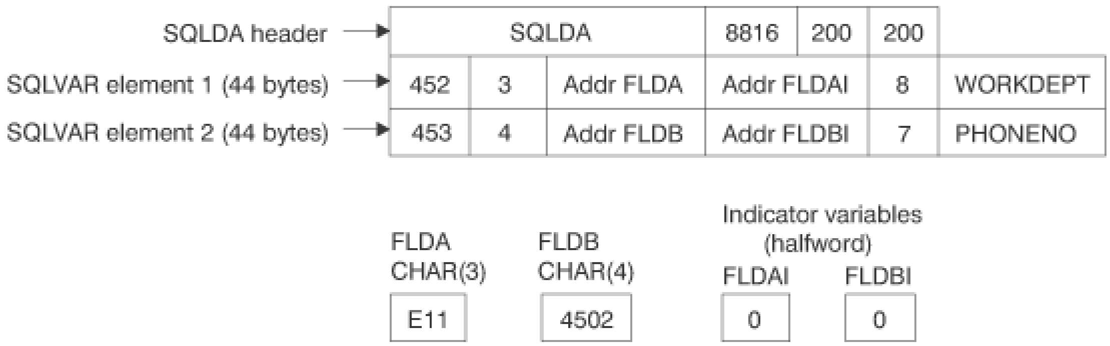


Figure 34. SQL descriptor area after executing FETCH

The following table describes the values in the descriptor area.

Table 91. Values inserted in the SQLDA

Value	Field	Description
SQLDA	SQLDAID	An "eye-catcher"
8816	SQLDABC	The size of the SQLDA in bytes (16 + 44 * 200)
200	SQLN	The number of occurrences of SQLVAR, set by the program
200	SQLD	The number of occurrences of SQLVAR actually used by the DESCRIBE statement
452	SQLTYPE	The value of SQLTYPE in the first occurrence of SQLVAR. It indicates that the first column contains fixed-length character strings, and does not allow nulls.
3	SQLLEN	The length attribute of the column
Undefined or CCSID value	SQLDATA	Bytes 3 and 4 contain the CCSID of a string column. Undefined for other types of columns.
Undefined	SQLIND	
8	SQLNAME	The number of characters in the column name
WORKDEPT	SQLNAME+2	The column name of the first column

Putting storage addresses in the SQLDA

After analyzing the description of each column, your program must replace the content of each SQLDATA field with the address of a storage area large enough to hold values from that column. Similarly, for every column that allows nulls, the program must replace the content of the SQLIND field. The content must be the address of a halfword that you can use as an indicator variable for the column. The program can acquire storage for this purpose, of course, but the storage areas used do not have to be contiguous.

Figure 33 on page 508 shows the content of the descriptor area before the program obtains any rows of the result table. Addresses of fields and indicator variables are already in the SQLVAR.

Changing the CCSID for retrieved data

All Db2 string data has an encoding scheme and CCSID associated with it. When you select string data from a table, the selected data generally has the same encoding scheme and CCSID as the table. If the application uses some method, such as issuing the DECLARE VARIABLE statement, to change the CCSID of the selected data, the data is converted from the CCSID of the table to the CCSID that is specified by the application.

You can set the default application encoding scheme for a plan or package by specifying the value in the APPLICATION ENCODING field of the panel DEFAULTS FOR BIND PACKAGE or DEFAULTS FOR

BIND PLAN. The default application encoding scheme for the Db2 subsystem is the value that was specified in the APPLICATION ENCODING field of installation panel DSNTIPF.

If you want to retrieve the data in an encoding scheme and CCSID other than the default values, you can use one of the following techniques:

- For dynamic SQL, set the CURRENT APPLICATION ENCODING SCHEME special register before you execute the SELECT statements. For example, to set the CCSID and encoding scheme for retrieved data to the default CCSID for Unicode, execute this SQL statement:

  ```
  EXEC SQL SET CURRENT APPLICATION ENCODING SCHEME ='UNICODE';
  ```

  The initial value of this special register is the application encoding scheme that is determined by the BIND option.
- For static and dynamic SQL statements that use host variables and host-variable arrays, use the DECLARE VARIABLE statement to associate CCSIDs with the host variables into which you retrieve the data. See "Setting the CCSID for host variables" on page 481 for information about this technique.
- For static and dynamic SQL statements that use a descriptor, set the CCSID for the retrieved data in the SQLDA. The following text describes that technique.

To change the encoding scheme for SQL statements that use a descriptor, set up the SQLDA, and then make these additional changes to the SQLDA:

1. Put the character + in the sixth byte of field SQLDAID.
2. For each SQLVAR entry:
   a. Set the length field of SQLNAME to 8.
   b. Set the first two bytes of the data field of SQLNAME to X'0000'.
   c. Set the third and fourth bytes of the data field of SQLNAME to the CCSID, in hexadecimal, in which you want the results to display, or to X'0000'. X'0000' indicates that Db2 should use the default CCSID If you specify a nonzero CCSID, it must meet one of the following conditions:
      - A row in catalog table SYSSTRINGS has a matching value for OUTCCSID.
      - The Unicode conversion services support conversion to that CCSID. See Building and using Dynamic Link Libraries (DLLs) (XL C/C++ Programming Guide) for information about the conversions supported.

      If you are modifying the CCSID to retrieve the contents of an ASCII, EBCDIC, or Unicode table on a Db2 for z/OS system, and you previously executed a DESCRIBE statement on the SELECT statement that you are using to retrieve the data, the SQLDATA fields in the SQLDA that you used for the DESCRIBE contain the ASCII or Unicode CCSID for that table. To set the data portion of the SQLNAME fields for the SELECT, move the contents of each SQLDATA field in the SQLDA from the DESCRIBE to each SQLNAME field in the SQLDA for the SELECT. If you are using the same SQLDA for the DESCRIBE and the SELECT, be sure to move the contents of the SQLDATA field to SQLNAME before you modify the SQLDATA field for the SELECT.

   For REXX, you set the CCSID in the *stem.n.*SQLUSECCSID field instead of setting the SQLDAID and SQLNAME fields.

For example, suppose that the table that contains WORKDEPT and PHONENO is defined with CCSID ASCII. To retrieve data for columns WORKDEPT and PHONENO in ASCII CCSID 437 (X'01B5'), change the SQLDA as shown in the following figure.

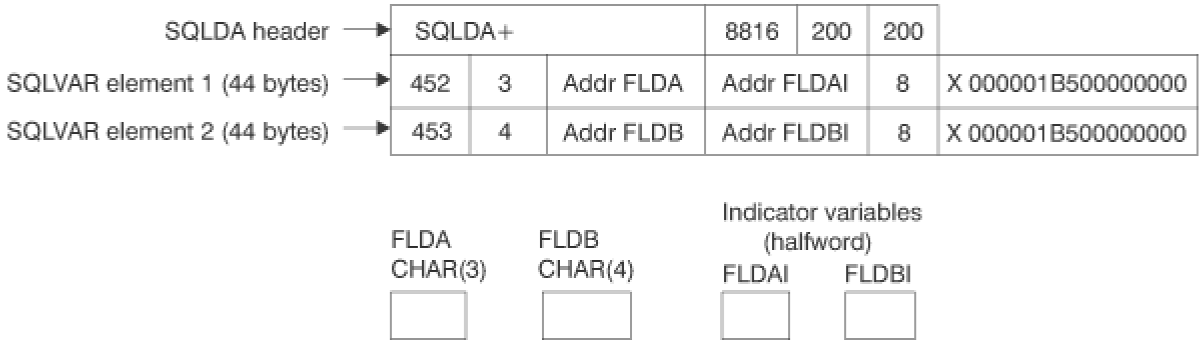


Figure 35. SQL descriptor area for retrieving data in ASCII CCSID 437

Specifying that DESCRIBE use column labels in the SQLNAME field

By default, DESCRIBE describes each column in the SQLNAME field by the column name. You can tell it to use column labels instead.

Restriction: You cannot use column labels with set operators (UNION, INTERSECT, and EXCEPT).

To specify that DESCRIBE use column labels in the SQLNAME field, specify one of the following options when you issue the DESCRIBE statement:

USING LABELS
Specifies that SQLNAME is to contain labels. If a column has no label, SQLNAME contains nothing.

USING ANY
Specifies that SQLNAME is to contain labels wherever they exist. If a column has no label, SQLNAME contains the column name.

USING BOTH
Specifies that SQLNAME is to contain both labels and column names, when both exist.

In this case, FULSQLDA must contain a second set of occurrences of SQLVAR. The first set contains descriptions of all the columns with column names; the second set contains descriptions with column labels.

If you choose this option, perform the following actions:

- Allocate a longer SQLDA for the second DESCRIBE statement ((16 + SQLD * 88 bytes) instead of (16 + SQLD * 44))
- Put double the number of columns (SLQD * 2) in the SQLN field of the second SQLDA.

These actions ensure that enough space is available. Otherwise, if not enough space is available, DESCRIBE does not enter descriptions of any of the columns.

```
EXEC SQL
  DESCRIBE STMT INTO :FULSQLDA USING LABELS;
```

Some columns, such as those derived from functions or expressions, have neither name nor label; SQLNAME contains nothing for those columns. For example, if you use a UNION to combine two columns that do not have the same name and do not use a label, SQLNAME contains a string of length zero.

Describing tables with LOB and distinct type columns

In general, the steps that you perform when you prepare an SQLDA to select rows from a table with LOB and distinct type columns are similar to the steps that you perform if the table has no columns of this type. The only difference is that you need to analyze some additional fields in the SQLDA for LOB or distinct type columns.

For example, Suppose that you want to execute this SELECT statement:

```
SELECT USER, A_DOC FROM DOCUMENTS;
```

The USER column cannot contain nulls and is of distinct type ID, defined like this:

```
CREATE DISTINCT TYPE SCHEMA1.ID AS CHAR(20);
```

The A_DOC column can contain nulls and is of type CLOB(1M).

The result table for this statement has two columns, but you need four SQLVAR occurrences in your SQLDA because the result table contains a LOB type and a distinct type. Suppose that you prepare and describe this statement into FULSQLDA, which is large enough to hold four SQLVAR occurrences. FULSQLDA looks like the following figure .

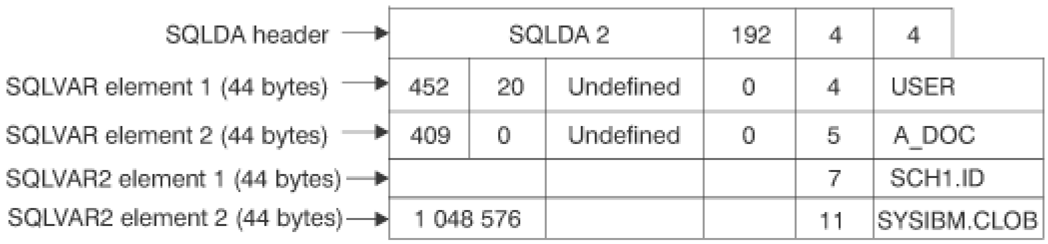


Figure 36. SQL descriptor area after describing a CLOB and distinct type

The next steps are the same as for result tables without LOBs or distinct types:

1. Analyze each SQLVAR description to determine the maximum amount of space you need for the column value.

   For a LOB type, retrieve the length from the SQLLONGL field instead of the SQLLEN field.

2. Derive the address of some storage area of the required size.

   For a LOB data type, you also need a 4-byte storage area for the length of the LOB data. You can allocate this 4-byte area at the beginning of the LOB data or in a different location.

3. Put this address in the SQLDATA field.

   For a LOB data type, if you allocated a separate area to hold the length of the LOB data, put the address of the length field in SQLDATAL. If the length field is at beginning of the LOB data area, put 0 in SQLDATAL. When you use a file reference variable for a LOB column, the indicator variable indicates whether the data in the file is null, not whether the data to which SQLDATA points is null.

4. If the SQLTYPE field indicates that the value can be null, the program must also put the address of an indicator variable in the SQLIND field.

The following figure shows the contents of FULSQLDA after you enter pointers to the storage locations.

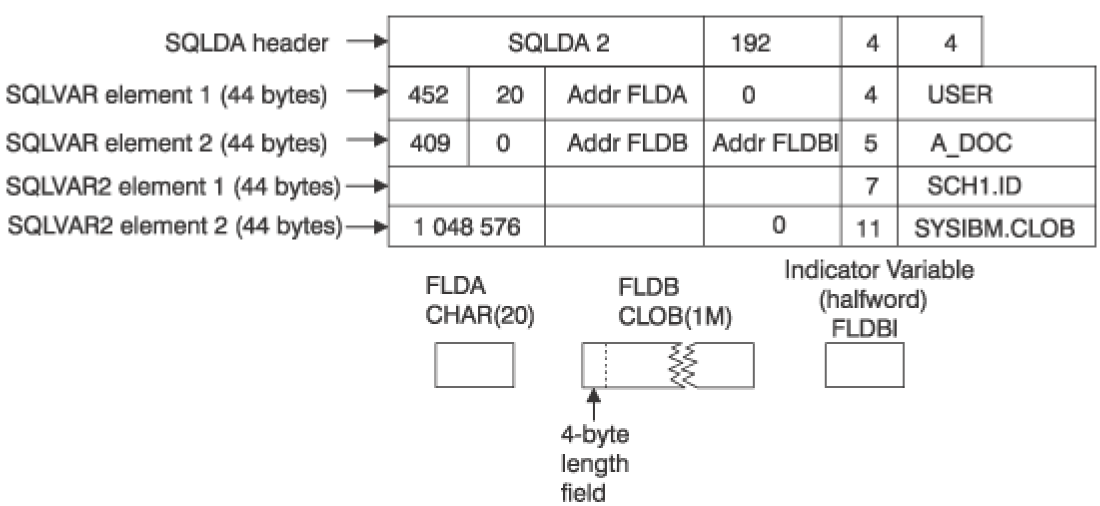


Figure 37. SQL descriptor area after analyzing CLOB and distinct type descriptions and acquiring storage

The following figure shows the contents of FULSQLDA after you execute a FETCH statement.

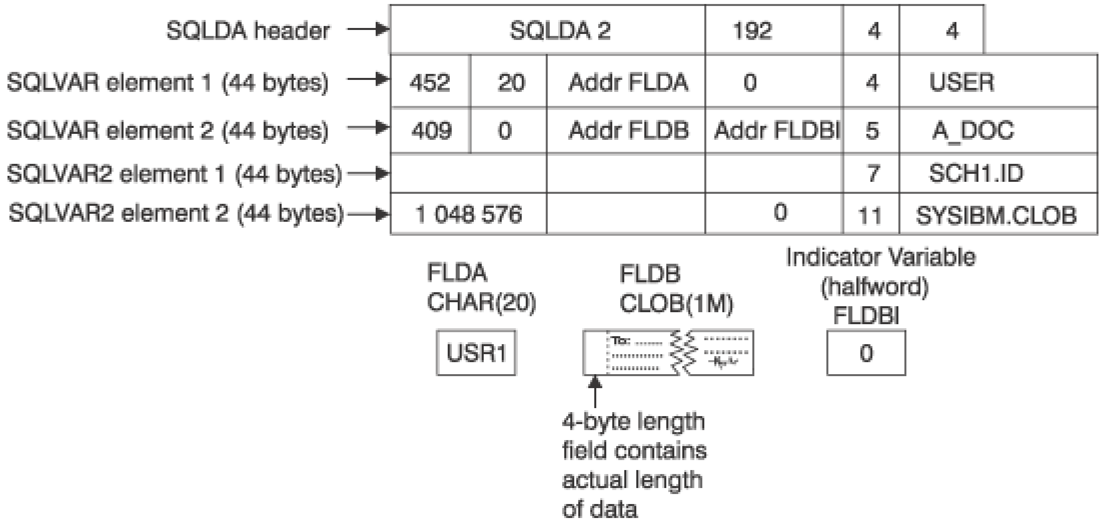


Figure 38. SQL descriptor area after executing FETCH on a table with CLOB and distinct type columns

Setting an XML host variable in an SQLDA

Instead of specifying host variables to store XML values from a table, you can create an SQLDA to point to the data areas where Db2 puts the retrieved data. The SQLDA needs to describe the data type for each data area.

To set an XML host variable in an SQLDA:

1. Allocate an appropriate SQLDA.
2. Issue a DESCRIBE statement for the SQL statement whose result set you want to store. The DESCRIBE statement populates the SQLDA based on the column definitions. In the SQLDA, an SQLVAR entry is populated for each column in the result set. (Multiple SQLVAR entries are populated for LOB columns and columns with distinct types.) For columns of type XML the associated SQLVAR entry is populated as follows:

Table 92. SQLVAR field values for XML columns

SQLVAR field	Value for an XML column
sqltype SQLTYPE	988 for a column that is not nullable or 989 for a nullable column
sqllen SQLLEN	0
sqldata SQLDATA	0
sqlind SQLIND	0
sqlname SQLNAME	The unqualified name or label of the column

3. Check the SQLTYPE field of each SQLVAR entry. If the SQLTYPE field is 988 or 989, the column in the result set is an XML column.
4. For each XML column, make the following changes to the associated SQLVAR entry:

a. Change the SQLTYPE field to indicate the data type of the host variable to receive the XML data. You can retrieve the XML data into a host variable of type XML AS BLOB, XML AS CLOB, or XML AS DBCLOB, or a compatible string data type.

   If the target host variable type is XML AS BLOB, XML AS CLOB, or XML AS DBCLOB, set the SQLTYPE field to one of the following values:

   404
   XML AS BLOB

   405
   nullable XML AS BLOB

   408
   XML AS CLOB

   409
   nullable XML AS CLOB

   412
   XML AS DBCLOB

   413
   nullable XML AS DBCLOB

   If the target host variable type is a string data type, set the SQLTYPE field to a valid string value.

   Restriction: You cannot use the XML type (988/989) as a target host variable type.

b. If the target host variable type is XML AS BLOB, XML AS CLOB, or XML AS DBCLOB, change the first two bytes in the SQLNAME field to X'0000' and the fifth and sixth bytes to X'0100'. These bytes indicate that the value to be received is an XML value.

5. Populate the extended SQLVAR fields for each XML column as you would for a LOB column, as indicated in the following table.

Table 93. Fields for an extended SQLVAR entry for an XML host variable

SQLVAR field	Value for an XML host variable
len.sqllonglen SQLLONGL SQLLONGLEN	length attribute for the XML host variable
*	Reserved
sqldatalen SQLDATAL SQLDATALEN	pointer to the length of the XML host variable
sqldatatype_name SQLTNAME SQLDATATYPENAME	not used

You can now use the SQLDA to retrieve the XML data into a host variable of type XML AS BLOB, XML AS CLOB, or XML AS DBCLOB, or a compatible string data type.

Executing a varying-list SELECT statement dynamically

You can easily retrieve rows of the result table using a varying-list SELECT statement. The statements differ only a little from those for the fixed-list example.

1. Open the cursor. If the SELECT statement contains no parameter marker, this step is simple enough. For example:

```
EXEC SQL OPEN C1;
```

2. Fetch rows from the result table. This statement differs from the corresponding one for the case of a fixed-list select. Write:

```
EXEC SQL
  FETCH C1 USING DESCRIPTOR :FULSQLDA;
```

The key feature of this statement is the clause USING DESCRIPTOR :FULSQLDA. That clause names an SQL descriptor area in which the occurrences of SQLVAR point to other areas. Those other areas receive the values that FETCH returns. It is possible to use that clause only because you previously set up FULSQLDA to look like Figure 32 on page 508.

Figure 34 on page 509 shows the result of the FETCH. The data areas identified in the SQLVAR fields receive the values from a single row of the result table.

Successive executions of the same FETCH statement put values from successive rows of the result table into these same areas.

3. Close the cursor. This step is the same as for the fixed-list case. When no more rows need to be processed, execute the following statement:

```
EXEC SQL CLOSE C1;
```

When COMMIT ends the unit of work containing OPEN, the statement in STMT reverts to the unprepared state. Unless you defined the cursor using the WITH HOLD option, you must prepare the statement again before you can reopen the cursor.

Executing arbitrary statements with parameter markers

Consider, as an example, a program that executes dynamic SQL statements of several kinds, including varying-list SELECT statements, any of which might contain a variable number of parameter markers. This program might present your users with lists of choices: choices of operation (update, select, delete); choices of table names; choices of columns to select or update. The program also enables the users to enter lists of employee numbers to apply to the chosen operation. From this, the program constructs SQL statements of several forms, one of which looks like this:

```
SELECT .... FROM DSN8B10.EMP
  WHERE EMPNO IN (?,?,?,...?);
```

The program then executes these statements dynamically.

When the number and types of parameters are known

In the preceding example, you do not know in advance the number of parameter markers, and perhaps the kinds of parameter they represent. You can use techniques described previously if you know the number and types of parameters, as in the following examples:

- If the SQL statement **is not** SELECT, name a list of host variables in the EXECUTE statement:

```
WRONG:     EXEC SQL EXECUTE STMT;

RIGHT:     EXEC SQL EXECUTE STMT USING :VAR1, :VAR2, :VAR3;
```

- If the SQL statement **is** SELECT, name a list of host variables in the OPEN statement:

```
WRONG:     EXEC SQL OPEN C1;

RIGHT:     EXEC SQL OPEN C1 USING :VAR1, :VAR2, :VAR3;
```

In **both** cases, the number and types of host variables named must agree with the number of parameter markers in STMT and the types of parameter they represent. The first variable (VAR1 in the examples) must have the type expected for the first parameter marker in the statement, the second variable must have the type expected for the second marker, and so on. There must be at least as many variables as parameter markers.

When the number and types of parameters are not known

When you do not know the number and types of parameters, you can adapt the SQL descriptor area. Your program can include an unlimited number of SQLDAs, and you can use them for different purposes. Suppose that an SQLDA, arbitrarily named DPARM, describes a set of parameters.

The structure of DPARM is the same as that of any other SQLDA. The number of occurrences of SQLVAR can vary, as in previous examples. In this case, every parameter marker must have one SQLVAR. Each occurrence of SQLVAR describes one host variable that replaces one parameter marker at run time. Db2 replaces the parameter markers when a non-SELECT statement executes or when a cursor is opened for a SELECT statement.

You must enter certain fields in DPARM **before** using EXECUTE or OPEN; you can ignore the other fields.

Field
: **Use when describing host variables for parameter markers**

SQLDAID
: The seventh byte indicates whether more than one SQLVAR entry is used for each parameter marker. If this byte is not blank, at least one parameter marker represents a distinct type or LOB value, so the SQLDA has more than one set of SQLVAR entries.

  You do not set this field for a REXX SQLDA.

SQLDABC
: The length of the SQLDA, which is equal to SQLN * 44 + 16. You do not set this field for a REXX SQLDA.

SQLN
: The number of occurrences of SQLVAR allocated for DPARM. You do not set this field for a REXX SQLDA.

SQLD
: The number of occurrences of SQLVAR actually used. This number must not be less than the number of parameter markers. In each occurrence of SQLVAR, put information in the following fields: SQLTYPE, SQLLEN, SQLDATA, SQLIND.

SQLTYPE
: The code for the type of variable, and whether it allows nulls.

SQLLEN
: The length of the host variable.

SQLDATA
: The address of the host variable.

  For REXX, this field contains the value of the host variable.

SQLIND
: The address of an indicator variable, if needed.

  For REXX, this field contains a negative number if the value in SQLDATA is null.

SQLNAME
: Ignore.

Using the SQLDA with EXECUTE or OPEN

To indicate that the SQLDA called DPARM describes the host variables substituted for the parameter markers at run time, use a USING DESCRIPTOR clause with EXECUTE or OPEN.

- For a non-SELECT statement, write:

```
EXEC SQL EXECUTE STMT USING DESCRIPTOR :DPARM;
```

- For a SELECT statement, write:

```
EXEC SQL OPEN C1 USING DESCRIPTOR :DPARM;
```

How bind options REOPT(ALWAYS), REOPT(AUTO) and REOPT(ONCE) affect dynamic SQL

When you specify the bind option REOPT(ALWAYS), Db2 reoptimizes the access path at run time for SQL statements that contain host variables, parameter markers, or special registers. The option REOPT(ALWAYS) has the following effects on dynamic SQL statements:

- When you specify the option REOPT(ALWAYS), Db2 automatically uses DEFER(PREPARE), which means that Db2 waits to prepare a statement until it encounters an OPEN or EXECUTE statement.
- When you execute a DESCRIBE statement and then an EXECUTE statement on a non-SELECT statement, Db2 prepares the statement twice: Once for the DESCRIBE statement and once for the EXECUTE statement. Db2 uses the values in the input variables only during the second PREPARE. These multiple PREPAREs can cause performance to degrade if your program contains many dynamic non-SELECT statements. To improve performance, consider putting the code that contains those statements in a separate package and then binding that package with the option REOPT(NONE).
- If you execute a DESCRIBE statement before you open a cursor for that statement, Db2 prepares the statement twice. If, however, you execute a DESCRIBE statement after you open the cursor, Db2 prepares the statement only once. To improve the performance of a program bound with the option REOPT(ALWAYS), execute the DESCRIBE statement **after** you open the cursor. To prevent an automatic DESCRIBE before a cursor is opened, do not use a PREPARE statement with the INTO clause.
- If you use predictive governing for applications bound with REOPT(ALWAYS), Db2 does not return a warning SQLCODE when dynamic SQL statements exceed the predictive governing warning threshold. Db2 does return an error SQLCODE when dynamic SQL statements exceed the predictive governing error threshold. Db2 returns the error SQLCODE for an EXECUTE or OPEN statement.

When you specify the bind option REOPT(AUTO), Db2 optimizes the access path for SQL statements at the first EXECUTE or OPEN. Each time a statement is executed, Db2 determines if a new access path is needed to improve the performance of the statement. If a new access path will improve the performance, Db2 generates one. The option REOPT(AUTO) has the following effects on dynamic SQL statements:

- When you specify the bind option REOPT(AUTO), Db2 optimizes the access path for SQL statements at the first EXECUTE or OPEN. Each time a statement is executed, Db2 determines if a new access path is needed to improve the performance of the statement. If a new access path will improve the performance, Db2 generates one.
- When you specify the option REOPT(ONCE), Db2 automatically uses DEFER(PREPARE), which means that Db2 waits to prepare a statement until it encounters an OPEN or EXECUTE statement.
- When Db2 prepares a statement using REOPT(AUTO), it saves the access path in the dynamic statement cache. This access path is used each time the statement is run, until Db2 determines that a new access path is needed to improve the performance or the statement that is in the cache is invalidated (or removed from the cache) and needs to be rebound.
- The DESCRIBE statement has the following effects on dynamic statements that are bound with REOPT(AUTO):
  - When you execute a DESCRIBE statement before an EXECUTE statement on a non-SELECT statement, Db2 prepares the statement an extra time if it is not already saved in the cache: Once for the DESCRIBE statement and once for the EXECUTE statement. Db2 uses the values of the input variables only during the second time the statement is prepared. It then saves the statement in the cache. If you execute a DESCRIBE statement before an EXECUTE statement on a non-SELECT statement that has already been saved in the cache, Db2 will always prepare the non-SELECT statement for the DESCRIBE statement, and will prepare the statement again on EXECUTE only if Db2 determines that a new access path different from the one already saved in the cache can improve the performance.
  - If you execute DESCRIBE on a statement before you open a cursor for that statement, Db2 always prepares the statement on DESCRIBE. However, Db2 will not prepare the statement again on OPEN if the statement has already been saved in the cache and Db2 does not think that a new access path is needed at OPEN time. If you execute DESCRIBE on a statement after you open a

cursor for that statement, Db2 prepared the statement only once if it is not already saved in the cache. If the statement is already saved in the cache and you execute DESCRIBE after you open a cursor for that statement, Db2 does not prepare the statement, it used the statement that is saved in the cache.

- If you use predictive governing for applications that are bound with REOPT(AUTO), Db2 does not return a warning SQLCODE when dynamic SQL statements exceed the predictive governing warning threshold. Db2 does return an error SQLCODE when dynamic SQL statements exceed the predictive governing error threshold. Db2 returns the error SQLCODE for an EXECUTE or OPEN statement.

When you specify the bind option REOPT(ONCE), Db2 optimizes the access path only once, at the first EXECUTE or OPEN, for SQL statements that contain host variables, parameter markers, or special registers. The option REOPT(ONCE) has the following effects on dynamic SQL statements:

- When you specify the option REOPT(ONCE), Db2 automatically uses DEFER(PREPARE), which means that Db2 waits to prepare a statement until it encounters an OPEN or EXECUTE statement.
- When Db2 prepares a statement using REOPT(ONCE), it saves the access path in the dynamic statement cache. This access path is used each time the statement is run, until the statement that is in the cache is invalidated (or removed from the cache) and needs to be rebound.
- The DESCRIBE statement has the following effects on dynamic statements that are bound with REOPT(ONCE):
  - When you execute a DESCRIBE statement before an EXECUTE statement on a non-SELECT statement, Db2 prepares the statement twice if it is not already saved in the cache: Once for the DESCRIBE statement and once for the EXECUTE statement. Db2 uses the values of the input variables only during the second time the statement is prepared. It then saves the statement in the cache. If you execute a DESCRIBE statement before an EXECUTE statement on a non-SELECT statement that has already been saved in the cache, Db2 prepares the non-SELECT statement only for the DESCRIBE statement.
  - If you execute DESCRIBE on a statement **before** you open a cursor for that statement, Db2 always prepares the statement on DESCRIBE. However, Db2 will not prepare the statement again on OPEN if the statement has already been saved in the cache. If you execute DESCRIBE on a statement **after** you open a cursor for that statement, Db2 prepared the statement only once if it is not already saved in the cache. If the statement is already saved in the cache and you execute DESCRIBE after you open a cursor for that statement, Db2 does not prepare the statement, it used the statement that is saved in the cache.

  To improve the performance of a program that is bound with REOPT(ONCE), execute the DESCRIBE statement after you open a cursor. To prevent an automatic DESCRIBE before a cursor is opened, do not use a PREPARE statement with the INTO clause.
- If you use predictive governing for applications that are bound with REOPT(ONCE), Db2 does not return a warning SQLCODE when dynamic SQL statements exceed the predictive governing warning threshold. Db2 does return an error SQLCODE when dynamic SQL statements exceed the predictive governing error threshold. Db2 returns the error SQLCODE for an EXECUTE or OPEN statement.

Related concepts

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements

You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related reference
DESCRIBE OUTPUT (Db2 SQL)
SQL descriptor area (SQLDA) (Db2 SQL)
SQLTYPE and SQLLEN (Db2 SQL)
The SQLDA Header (Db2 SQL)

Dynamically executing an SQL statement by using EXECUTE IMMEDIATE

In certain situations, you might want your program to prepare and dynamically execute a statement immediately after reading it.

About this task

Suppose that you design a program to read SQL DELETE statements, similar to these, from a terminal:

```
DELETE FROM DSN8B10.EMP WHERE EMPNO = '000190'
DELETE FROM DSN8B10.EMP WHERE EMPNO = '000220'
```

After reading a statement, the program is to run it immediately.

Recall that you must prepare (precompile and bind) static SQL statements before you can use them. You cannot prepare dynamic SQL statements in advance. The SQL statement EXECUTE IMMEDIATE causes an SQL statement to prepare and execute, dynamically, at run time.

Before you prepare and execute an SQL statement, you can read it into a host variable. If the maximum length of the SQL statement is 32 KB, declare the host variable as a character or graphic host variable according to the following rules for the host languages:

- In assembler, PL/I, COBOL and C, you must declare a string host variable as a varying-length string.
- In Fortran, it must be a fixed-length string variable.

If the length is greater than 32 KB, you must declare the host variable as a CLOB or DBCLOB, and the maximum is 2 MB.

Examples

Example: Using a varying-length character host variable
This excerpt is from a C program that reads a DELETE statement into the host variable *dstring* and executes the statement:

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
    struct VARCHAR {
      short len;
      char s[40];
      } dstring;
EXEC SQL END DECLARE SECTION;
...
/* Read a DELETE statement into the host variable dstring. */
gets(dstring);
EXEC SQL EXECUTE IMMEDIATE :dstring;
...
```

EXECUTE IMMEDIATE causes the DELETE statement to be prepared and executed immediately.

Declaring a CLOB or DBCLOB host variable
You declare CLOB and DBCLOB host variables according to certain rules.

The precompiler generates a structure that contains two elements, a 4-byte length field and a data field of the specified length. The names of these fields vary depending on the host language:

- In PL/I, assembler, and Fortran, the names are *variable*_LENGTH and *variable*_DATA.
- In COBOL, the names are *variable*–LENGTH and *variable*–DATA.
- In C, the names are *variable*.LENGTH and *variable*.DATA.

Example: Using a CLOB host variable

This excerpt is from a C program that copies an UPDATE statement into the host variable *string1* and executes the statement:

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
    SQL TYPE IS CLOB(4k) string1;
EXEC SQL END DECLARE SECTION;
...
/* Copy a statement into the host variable string1. */
strcpy(string1.data, "UPDATE DSN8610.EMP SET SALARY = SALARY * 1.1");
string1.length = 44;
EXEC SQL EXECUTE IMMEDIATE :string1;
...
```

EXECUTE IMMEDIATE causes the UPDATE statement to be prepared and executed immediately.

Related concepts

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements
You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Dynamically executing an SQL statement by using PREPARE and EXECUTE

As an alternative to executing an SQL statement immediately after it is read, you can prepare and execute the SQL statement in two steps. This two-step method is useful when you need to execute an SQL statement multiple times with different values.

About this task

Suppose that you want to execute DELETE statements repeatedly using a list of employee numbers. Consider how you would do it if you could write the DELETE statement as a static SQL statement:

```
< Read a value for EMP from the list. >
DO UNTIL (EMP = 0);
  EXEC SQL
    DELETE FROM DSN8B10.EMP WHERE EMPNO = :EMP ;
  < Read a value for EMP from the list. >
END;
```

The loop repeats until it reads an EMP value of 0.

If you know in advance that you will use only the DELETE statement and only the table DSN8B10.EMP, you can use the more efficient static SQL. Suppose further that several different tables have rows that are identified by employee numbers, and that users enter a table name as well as a list of employee numbers to delete. Although variables can represent the employee numbers, they cannot represent the table name, so you must construct and execute the entire statement dynamically.

Procedure

To construct and execute statements dynamically your program must now do these things differently:

- Use parameter markers instead of host variables.

  Dynamic SQL statements cannot use host variables. Therefore, you cannot dynamically execute an SQL statement that contains host variables. Instead, substitute a *parameter marker*, indicated by a question mark (?), for each host variable in the statement.

  You can indicate to Db2 that a parameter marker represents a host variable of a certain data type by specifying the parameter marker as the argument of a CAST specification. When the statement executes, Db2 converts the host variable to the data type in the CAST specification. A parameter marker that you include in a CAST specification is called a *typed* parameter marker. A parameter marker without a CAST specification is called an *untyped* parameter marker.

  Recommendation: Because Db2 can evaluate an SQL statement with typed parameter markers more efficiently than a statement with untyped parameter markers, use typed parameter markers whenever possible. Under certain circumstances you must use typed parameter markers.

  For example, suppose that you want to prepare this statement:

  ```
  DELETE FROM DSN8B10.EMP WHERE EMPNO = :EMP;
  ```

  You need to prepare a string like this:

  ```
  DELETE FROM DSN8B10.EMP WHERE EMPNO = CAST(? AS CHAR(6))
  ```

  You associate host variable :EMP with the parameter marker when you execute the prepared statement. Suppose that S1 is the prepared statement. Then the EXECUTE statement looks like this:

  ```
  EXECUTE S1 USING :EMP;
  ```

- Use the PREPARE statement.

  Before you prepare an SQL statement, you can assign it to a host variable. If the length of the statement is greater than 32 KB, you must declare the host variable as a CLOB or DBCLOB.

  You can think of PREPARE and EXECUTE as an EXECUTE IMMEDIATE done in two steps. The first step, PREPARE, turns a character string into an SQL statement, and then assigns it a name of your choosing.

  For example, assume that the character host variable :DSTRING has the value “DELETE FROM DSN8B10.EMP WHERE EMPNO = ?”. To prepare an SQL statement from that string and assign it the name S1, write:

  ```
  EXEC SQL PREPARE S1 FROM :DSTRING;
  ```

  The prepared statement still contains a parameter marker, for which you must supply a value when the statement executes. After the statement is prepared, the table name is fixed, but the parameter marker enables you to execute the same statement many times with different values of the employee number.

- Use EXECUTE instead of EXECUTE IMMEDIATE.

  The EXECUTE statement executes a prepared SQL statement by naming a list of one or more host variables, one or more host-variable arrays, or a host structure. This list supplies values for all of the parameter markers.

  After you prepare a statement, you can execute it many times within the same unit of work. In most cases, COMMIT or ROLLBACK destroys statements prepared in a unit of work. Then, you must prepare them again before you can execute them again. However, if you declare a cursor for a dynamic statement and use the option WITH HOLD, a commit operation does not destroy the prepared statement if the cursor is still open. You can execute the statement in the next unit of work without preparing it again.

For example, to execute the prepared statement S1 just once, using a parameter value contained in the host variable :EMP, write:

```
EXEC SQL EXECUTE S1 USING :EMP;
```

Examples

Preparing and executing the example DELETE statement

```
< Read a value for EMP from the list. >
DO UNTIL (EMP = 0);
  EXEC SQL
    DELETE FROM DSN8B10.EMP WHERE EMPNO = :EMP ;
  < Read a value for EMP from the list. >
END;
```

You can now write an equivalent example for a dynamic SQL statement:

```
< Read a statement containing parameter markers into DSTRING.>
EXEC SQL PREPARE S1 FROM :DSTRING;
< Read a value for EMP from the list. >
DO UNTIL (EMPNO = 0);
  EXEC SQL EXECUTE S1 USING :EMP;
  < Read a value for EMP from the list. >
  END;
```

The PREPARE statement prepares the SQL statement and calls it S1. The EXECUTE statement executes S1 repeatedly, using different values for EMP.

Using more than one parameter marker

The prepared statement (S1 in the example) can contain more than one parameter marker. If it does, the USING clause of EXECUTE specifies a list of variables or a host structure. The variables must contain values that match the number and data types of parameters in S1 in the proper order. You must know the number and types of parameters in advance and declare the variables in your program, or you can use an SQLDA (SQL descriptor area).

Related concepts

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements
You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related tasks

Dynamically executing an SQL statement by using EXECUTE IMMEDIATE

In certain situations, you might want your program to prepare and dynamically execute a statement immediately after reading it.

Related reference

PREPARE (Db2 SQL)

Dynamically executing a data change statement

Dynamically executing data change statements with host-variable arrays is useful if you want to enter rows of data into different tables. It is also useful if you want to enter a different number of rows. The process is similar for both INSERT and MERGE statements.

About this task

For example, suppose that you want to repeatedly execute a multiple-row INSERT statement with a list of activity IDs, activity keywords, and activity descriptions that are provided by the user. You can use the following static SQL INSERT statement to insert multiple rows of data into the activity table:

```
EXEC SQL
  INSERT INTO DSN8B10.ACT
    VALUES (:hva_actno, :hva_actkwd, :hva_actdesc)
    FOR :num_rows ROWS;
```

However, if you want to enter the rows of data into different tables or enter different numbers of rows, you can construct the INSERT statement dynamically.

Procedure

To execute a data change statement dynamically, use one of the following methods:

- Use host-variable arrays that contain the data to be inserted, by completing the following actions in your program:
  a) Assign the appropriate INSERT or MERGE statement to a host variable. If needed, use the CAST specification to explicitly assign types to parameter markers that represent host-variable arrays. For the activity table, the following string contains an INSERT statement that is to be prepared:

  ```
  INSERT INTO DSN8B10.ACT
    VALUES (CAST(? AS SMALLINT), CAST(? AS CHAR(6)), CAST(? AS VARCHAR(20)))
  ```

  b) Assign any attributes for the SQL statement to a host variable.

  c) Include a PREPARE statement for the SQL statement.

  d) Include an EXECUTE statement with the FOR *n* ROWS clause.

  Each host variable in the USING clause of the EXECUTE statement represents an array of values for the corresponding column of the target of the SQL statement. You can vary the number of rows without needing to prepare the SQL statement again.

  For example, the following code prepares and executes an INSERT statement:

  ```
  /* Copy the INSERT string into the host variable sqlstmt */
  strcpy(sqlstmt, "INSERT INTO DSN8B10.ACT VALUES (CAST(? AS SMALLINT),");
  strcat(sqlstmt, " CAST(? AS CHAR(6)), CAST(? AS VARCHAR(20)))");

  /* Copy the INSERT attributes into the host variable attrvar */
  strcpy(attrvar, "FOR MULTIPLE ROWS");

  /* Prepare and execute my_insert using the host-variable arrays */
  EXEC SQL PREPARE my_insert ATTRIBUTES :attrvar FROM :sqlstmt;
  EXEC SQL EXECUTE my_insert USING :hva1, :hva2, :hva3 FOR :num_rows ROWS;
  ```

- Use descriptor to describe the host-variable arrays that contain the data, by completing the following actions in your program:
  a) Set the following fields in the SQLDA structure to specify data types and other information about the host-variable arrays that contain the values to insert in your INSERT statement.

– SQLN
– SQLABC
– SQLD
– SQLVAR
– SQLNAME

Assume that your program includes the standard SQLDA structure declaration and declarations for the program variables that point to the SQLDA structure. For C application programs, the following example code sets the SQLDA fields:

```
strcpy(sqldaptr->sqldaid,"SQLDA");
sqldaptr->sqldabc = 192;   /* number of bytes of storage allocated
for the SQLDA */
sqldaptr->sqln = 4;                              /* number of SQLVAR
occurrences */
sqldaptr->sqld = 4;
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]));    /* Point
to first SQLVAR */
varptr->sqltype = 500;                                     /* data
type SMALLINT */
varptr->sqllen = 2;
varptr->sqldata = (char *) hva1;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 1); /* Point
to next SQLVAR */
varptr->sqltype = 452;                                      /* data
type CHAR(6) */
varptr->sqllen = 6;
varptr->sqldata = (char *) hva2;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 2); /* Point
to next SQLVAR */
varptr->sqltype = 448;                                  /* data type
VARCHAR(20) */
varptr->sqllen = 20;
varptr->sqldata = (char *) hva3;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14",varptr->sqlname.length);
```

The SQLDA structure has the following fields:

– SQLDABC indicates the number of bytes of storage that are allocated for the SQLDA. The storage includes a 16-byte header and 44 bytes for each SQLVAR field. The value is SQLN x 44 + 16, or 192 for this example.
– SQLN is the number of SQLVAR occurrences, plus one for use by Db2 for the host variable that contains the number *n* in the FOR *n* ROWS clause.
– SQLD is the number of variables in the SQLDA that are used by Db2 when processing the INSERT statement.
– An SQLVAR occurrence specifies the attributes of an element of a host-variable array that corresponds to a value provided for a target column of the INSERT. Within each SQLVAR:
  - SQLTYPE indicates the data type of the elements of the host-variable array.
  - SQLLEN indicates the length of a single element of the host-variable array.
  - SQLDATA points to the corresponding host-variable array. Assume that your program allocates the dynamic variable arrays hva1, hva2, and hva3.
  - SQLNAME has two parts: the LENGTH and the DATA. The LENGTH is 8. The first two bytes of the DATA field is X'0000'. Bytes 5 and 6 of the DATA field are a flag indicating whether the variable is an array or a FOR n ROWS value. Bytes 7 and 8 are a two-byte binary integer representation of the dimension of the array.

b) Assign the appropriate INSERT or MERGE statement to a host variable.
For example, the following string contains an INSERT statement that is to be prepared:

```
INSERT INTO DSN8B10.ACT VALUES (?, ?, ?)
```

c) Assign any attributes for the SQL statement to a host variable.

d) Include a PREPARE statement for the SQL statement.

e) Include an EXECUTE statement with the FOR *n* ROWS clause. The host variable in the USING clause of the EXECUTE statement names the SQLDA that describes the parameter markers in the INSERT statement.

For example, the following code prepares and executes an INSERT statement:

```
/* Copy the INSERT string into the host variable sqlstmt */
strcpy(sqlstmt, "INSERT INTO DSN8B10.ACT VALUES (?, ?, ?)");

/* Copy the INSERT attributes into the host variable attrvar */
strcpy(attrvar, "FOR MULTIPLE ROWS");

/* Prepare and execute my_insert using the descriptor */
EXEC SQL PREPARE my_insert ATTRIBUTES :attrvar FROM :sqlstmt;
EXEC SQL EXECUTE my_insert USING DESCRIPTOR :*sqldaptr FOR :num_rows ROWS;
```

Related concepts

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements
You can code SQL statements in a PL/I program wherever you can use executable statements.

Related tasks

Including dynamic SQL for varying-list SELECT statements in your program
A varying-list SELECT statement returns rows that contain an unknown number of values of unknown type. When you use this type of statement, you do not know in advance exactly what kinds of host variables you need to declare for storing the results.

Related reference

SQLTYPE and SQLLEN (Db2 SQL)

Dynamically executing a statement with parameter markers by using the SQLDA

Your program can get data type information about parameter markers by asking Db2 to set the fields in the SQLDA.

Before you begin

Before you dynamically execute a statement with parameter markers, allocate an SQLDA with enough instances of SQLVAR to represent all parameter markers in the SQL statement.

Procedure

To dynamically execute a statement with parameter markers by using the SQLDA:

1. Include in your program a DESCRIBE INPUT statement that specifies the prepared SQL statement and the name of an appropriate SQLDA.

Db2 puts the requested parameter marker information in the SQLDA.

2. Code the application in the same way as any other application in which you execute a prepared statement by using an SQLDA. First, obtain the addresses of the input host variables and their indicator variables and insert those addresses into the SQLDATA and SQLIND fields. Then, execute the prepared SQL statement.

Example

Suppose that you want to execute the following statement dynamically:

```
DELETE FROM DSN8B10.EMP WHERE EMPNO = ?
```

You can use the following code to set up an SQLDA, obtain parameter information by using the DESCRIBE INPUT statement, and execute the statement:

```
SQLDAPTR=ADDR(INSQLDA);           /* Get pointer to SQLDA         */
SQLDAID='SQLDA';                  /* Fill in SQLDA eye-catcher    */
SQLDABC=LENGTH(INSQLDA);          /* Fill in SQLDA length         */
SQLN=1;                           /* Fill in number of SQLVARs    */
SQLD=0;                           /* Initialize # of SQLVARs used */
DO IX=1 TO SQLN;                  /* Initialize the SQLVAR        */
  SQLTYPE(IX)=0;
  SQLLEN(IX)=0;
  SQLNAME(IX)='';
END;
SQLSTMT='DELETE FROM DSN8B10.EMP WHERE EMPNO = ?';
EXEC SQL PREPARE SQLOBJ FROM SQLSTMT;
EXEC SQL DESCRIBE INPUT SQLOBJ INTO :INSQLDA;
SQLDATA(1)=ADDR(HVEMP);           /* Get input data address       */
SQLIND(1)=ADDR(HVEMPIND);         /* Get indicator address        */
EXEC SQL EXECUTE SQLOBJ USING DESCRIPTOR :INSQLDA;
```

Related concepts

Assembler applications that issue SQL statements
You can code SQL statements in assembler programs wherever you can use executable statements.

C and C++ applications that issue SQL statements
You can code SQL statements in a C or C++ program wherever you can use executable statements.

COBOL applications that issue SQL statements
You can code SQL statements in certain COBOL program sections.

Fortran applications that issue SQL statements
You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

PL/I applications that issue SQL statements
You can code SQL statements in a PL/I program wherever you can use executable statements.

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

DESCRIBE INPUT (Db2 SQL)

Checking the execution of SQL statements

After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

About this task

You can check the execution of SQL statements in one of the following ways:

- By displaying specific fields in the SQLCA.
- By testing SQLCODE or SQLSTATE for specific values.
- By using the WHENEVER statement in your application program.
- By testing indicator variables to detect numeric errors.
- By using the GET DIAGNOSTICS statement in your application program to return all the condition information that results from the execution of an SQL statement.
- By calling DSNTIAR to display the contents of the SQLCA.

Related concepts

Arithmetic and conversion errors
You can track arithmetic and conversion errors by using indicator variables. An indicator variable contains a small integer value that indicates some information about the associated host variable.

Related tasks

Defining the SQL communications area, SQLSTATE, and SQLCODE in assembler
Assembler programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in C and C++
C and C++ programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL
COBOL programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in Fortran
Fortran programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I
PL/I programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX
When Db2 prepares a REXX program that contains SQL statements, Db2 automatically includes an SQLCA in the program.

Displaying SQLCA fields by calling DSNTIAR

If you use the SQLCA to check whether an SQL statement executed successfully, your program needs to read the data in the appropriate SQLCA fields. One easy way to read these fields is to use the assembler subroutine DSNTIAR.

Checking the execution of SQL statements by using the SQLCA

One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

About this task

If you use the SQLCA, include the necessary instructions to display information that is contained in the SQLCA in your application program. Alternatively, you can use the GET DIAGNOSTICS statement, which is an SQL standard, to diagnose problems.

- When Db2 processes an SQL statement, it places return codes that indicate the success or failure of the statement execution in SQLCODE and SQLSTATE.
- When Db2 processes a FETCH statement, and the FETCH is successful, the contents of SQLERRD(3) in the SQLCA is set to the number of returned rows.
- When Db2 processes a multiple-row FETCH statement, the contents of SQLCODE is set to +100 if the last row in the table has been returned with the set of rows.
- When Db2 processes an UPDATE, INSERT, or DELETE statement, and the statement execution is successful, the contents of SQLERRD(3) in the SQLCA is set to the number of rows that are updated, inserted, or deleted.
- When Db2 processes a TRUNCATE statement and the statement execution is successful, SQLERRD(3) in the SQLCA is set to -1. The number of rows that are deleted is not returned.
- If SQLWARN0 contains **W**, Db2 has set at least one of the SQL warning flags (SQLWARN1 through SQLWARNA):
  - SQLWARN1 contains **N** for non-scrollable cursors and **S** for scrollable cursors after an OPEN CURSOR or ALLOCATE CURSOR statement.
  - SQLWARN4 contains **I** for insensitive scrollable cursors, **S** for sensitive static scrollable cursors, and **D** for sensitive dynamic scrollable cursors, after an OPEN CURSOR or ALLOCATE CURSOR statement, or blank if the cursor is not scrollable.
  - SQLWARN5 contains a character value of **1** (read only), **2** (read and delete), or **4** (read, delete, and update) to indicate the operation that is allowed on the result table of the cursor.

Related tasks

Accessing data by using a rowset-positioned cursor
A rowset-positioned cursor is a cursor that can return one or more rows for a single fetch operation. The cursor is positioned on the set of rows that are to be fetched.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the SQL communications area, SQLSTATE, and SQLCODE in assembler
Assembler programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in C and C++
C and C++ programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL

COBOL programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in Fortran
Fortran programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I
PL/I programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX
When Db2 prepares a REXX program that contains SQL statements, Db2 automatically includes an SQLCA in the program.

Related reference

Description of SQLCA fields (Db2 SQL)

Displaying SQLCA fields by calling DSNTIAR

If you use the SQLCA to check whether an SQL statement executed successfully, your program needs to read the data in the appropriate SQLCA fields. One easy way to read these fields is to use the assembler subroutine DSNTIAR.

About this task

You should check for errors codes before you commit data, and handle the errors that they represent. The assembler subroutine DSNTIAR helps you to obtain a formatted form of the SQLCA and a text message based on the SQLCODE field of the SQLCA. You can retrieve this same message text by using the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement. Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR.

DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program. Each time you use DSNTIAR, it overwrites any previous messages in the message output area. You should move or print the messages before using DSNTIAR again, and before the contents of the SQLCA change, to get an accurate view of the SQLCA.

DSNTIAR expects the SQLCA to be in a certain format. If your application modifies the SQLCA format before you call DSNTIAR, the results are unpredictable.

DSNTIAR

The assembler subroutine DSNTIAR helps you to obtain a formatted form of the SQLCA and a text message that is based on the SQLCODE field of the SQLCA.

DSNTIAR can run either above or below the 16-MB line of virtual storage. The DSNTIAR object module that comes with Db2 has the attributes AMODE(31) and RMODE(ANY). At installation time, DSNTIAR links as AMODE(31) and RMODE(ANY). DSNTIAR runs in 31-bit mode if any of the following conditions is true:

- DSNTIAR is linked with other modules that also have the attributes AMODE(31) and RMODE(ANY).
- DSNTIAR is linked into an application that specifies the attributes AMODE(31) and RMODE(ANY) in its link-edit JCL.
- An application loads DSNTIAR.

When loading DSNTIAR from another program, be careful how you branch to DSNTIAR. For example, if the calling program is in 24-bit addressing mode and DSNTIAR is loaded above the 16-MB line, you cannot use the assembler BALR instruction or CALL macro to call DSNTIAR, because they assume that DSNTIAR is in 24-bit mode. Instead, you must use an instruction that is capable of branching into 31-bit mode, such as BASSM.

You can dynamically link (load) and call DSNTIAR directly from a language that does not handle 31-bit addressing. To do this, link a second version of DSNTIAR with the attributes AMODE(24) and RMODE(24) into another load module library. Alternatively, you can write an intermediate assembler language program that calls DSNTIAR in 31-bit mode and then call that intermediate program in 24-bit mode from your application.

For more information on the allowed and default AMODE and RMODE settings for a particular language, see the application programming guide for that language. For details on how the attributes AMODE and RMODE of an application are determined, see the linkage editor and loader user's guide for the language in which you have written the application.

Defining a message output area

If a program calls DSNTIAR, the program must allocate enough storage in the message output area to hold all of the message text that DSNTIAR returns.

About this task

You will probably need no more than 10 lines, 80-bytes each, for your message output area. An application program can have only one message output area.

You must define the message output area in VARCHAR format. In this varying character format, a 2-byte length field precedes the data. The length field indicates to DSNTIAR how many total bytes are in the output message area; the minimum length of the output area is 240-bytes.

The following figure shows the format of the message output area, where *length* is the 2-byte total length field, and the length of each line matches the logical record length (*lrecl*) you specify to DSNTIAR.

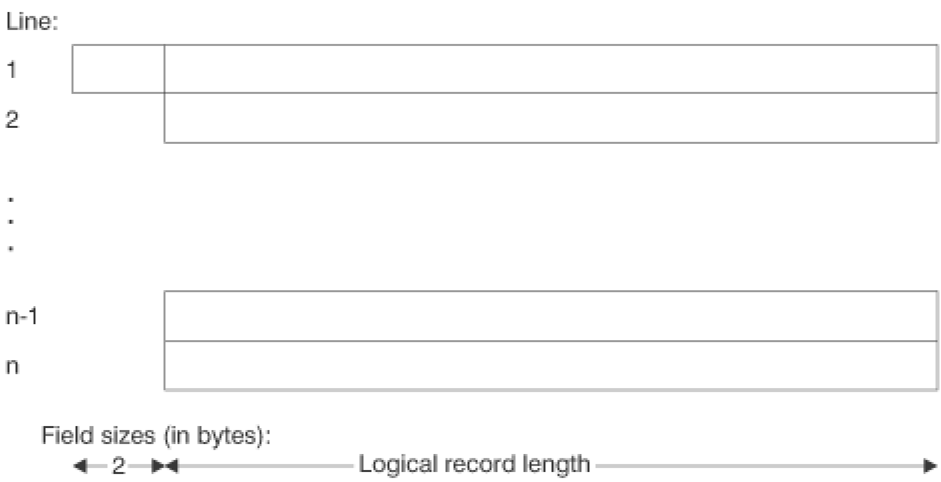


Figure 39. Format of the message output area

When you call DSNTIAR, you must name an SQLCA and an output message area in the DSNTIAR parameters. You must also provide the logical record length (*lrecl*) as a value between 72 and 240 bytes. DSNTIAR assumes the message area contains fixed-length records of length *lrecl*.

DSNTIAR places up to 10 lines in the message area. If the text of a message is longer than the record length you specify on DSNTIAR, the output message splits into several records, on word boundaries if possible. The split records are indented. All records begin with a blank character for carriage control. If you have more lines than the message output area can contain, DSNTIAR issues a return code of 4. A completely blank record marks the end of the message output area.

Possible return codes from DSNTIAR

The assembler subroutine DSNTIAR helps your program read the information in the SQLCA. The subroutine also returns its own return code.

Code
 Meaning

0
: Successful execution.

4
: More data available than could fit into the provided message area.

8
: Logical record length not between 72 and 240, inclusive.

12
: Message area not large enough. The message length was 240 or greater.

16
: Error in TSO message routine.

20
: Module DSNTIA1 could not be loaded.

24
: SQLCA data error.

A scenario for using DSNTIAR

You can use the assembler subroutine DSNTIAR to generate the error message text in the SQLCA.

Suppose you want your Db2 COBOL application to check for deadlocks and timeouts, and you want to make sure your cursors are closed before continuing. You use the statement WHENEVER SQLERROR to transfer control to an error routine when your application receives a negative SQLCODE.

In your error routine, you write a section that checks for SQLCODE -911 or -913. You can receive either of these SQLCODEs when a deadlock or timeout occurs. When one of these errors occurs, the error routine closes your cursors by issuing the statement:

```
EXEC SQL CLOSE cursor-name
```

An SQLCODE of 0 or -501 resulting from that statement indicates that the close was successful.

To use DSNTIAR to generate the error message text, first follow these steps:

1. Choose a logical record length (*lrecl*) of the output lines. For this example, assume *lrecl* is 72 (to fit on a terminal screen) and is stored in the variable named ERROR-TEXT-LEN.
2. Define a message area in your COBOL application. Assuming you want an area for up to 10 lines of length 72, you should define an area of 720 bytes, plus a 2-byte area that specifies the total length of the message output area.

   ```
   01  ERROR-MESSAGE.
           02  ERROR-LEN   PIC S9(4)  COMP VALUE +720.
           02  ERROR-TEXT  PIC X(72)  OCCURS 10 TIMES
                                      INDEXED BY ERROR-INDEX.
   77  ERROR-TEXT-LEN      PIC S9(9)  COMP VALUE +72.
   ```

   For this example, the name of the message area is ERROR-MESSAGE.
3. Make sure you have an SQLCA. For this example, assume the name of the SQLCA is SQLCA.

To display the contents of the SQLCA when SQLCODE is 0 or -501, call DSNTIAR after the SQL statement that produces SQLCODE 0 or -501:

```
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
```

You can then print the message output area just as you would any other variable. Your message might look like this:

```
DSNT408I SQLCODE = -501, ERROR:  THE CURSOR IDENTIFIED IN A FETCH OR
         CLOSE STATEMENT IS NOT OPEN
DSNT418I SQLSTATE   = 24501 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXERT SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -315  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFEC5'  X'00000000'  X'00000000'
```

```
        X'FFFFFFFF'  X'00000000'  X'00000000' SQL DIAGNOSTIC
        INFORMATION
```

Checking the execution of SQL statements by using SQLCODE and SQLSTATE

Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Procedure

You can declare SQLCODE and SQLSTATE (SQLCOD and SQLSTA in Fortran) as stand-alone host variables. If you specify the STDSQL(YES) precompiler option, these host variables receive the return codes, and you should not include an SQLCA in your program.

Portable applications should use SQLSTATE instead of SQLCODE, although SQLCODE values can provide additional Db2-specific information about an SQL error or warning.

An advantage to using the SQLCODE field is that it can provide more specific information than the SQLSTATE. Many of the SQLCODEs have associated tokens in the SQLCA that indicate, for example, which object incurred an SQL error. However, an SQL standard application uses only SQLSTATE.

SQLCODE

Db2 returns the following codes in SQLCODE:

- If SQLCODE = 0, execution was successful.
- If SQLCODE > 0, execution was successful with a warning.
- If SQLCODE < 0, execution was not successful.

SQLCODE 100 indicates that no data was found.

The meaning of SQLCODEs other than 0 and 100 varies with the particular product implementing SQL.

SQLSTATE

SQLSTATE enables an application program to check for errors in the same way for different IBM database management systems.

Related tasks

Defining the SQL communications area, SQLSTATE, and SQLCODE in assembler
Assembler programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in C and C++
C and C++ programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL
COBOL programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in Fortran
Fortran programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I
PL/I programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX

When Db2 prepares a REXX program that contains SQL statements, Db2 automatically includes an SQLCA in the program.

Related reference

SQLSTATE values and common error codes (Db2 Codes)

Checking the execution of SQL statements by using the WHENEVER statement

The WHENEVER statement causes Db2 to check the SQLCA and continue processing your program. If an error, exception, or warning occurs, Db2 branches to another area in your program. The condition handling area of your program can then examine the SQLCODE or SQLSTATE to react specifically to the error or exception.

About this task

The WHENEVER statement is not supported for REXX.

The WHENEVER statement enables you to specify what to do if a general condition is true. You can specify more than one WHENEVER statement in your program. When you do this, the first WHENEVER statement applies to all subsequent SQL statements in the source program until the next WHENEVER statement.

The WHENEVER statement looks like this:

```
EXEC SQL
  WHENEVER condition action
END-EXEC
```

The *condition* of the WHENEVER statement is one of these three values:

SQLWARNING
Indicates what to do when SQLWARN0 = W or SQLCODE contains a positive value other than 100. Db2 can set SQLWARN0 for several reasons—for example, if a column value is truncated when moved into a host variable. Your program might not regard this as an error.

SQLERROR
Indicates what to do when Db2 returns an error code as the result of an SQL statement (SQLCODE < 0).

NOT FOUND
Indicates what to do when Db2 cannot find a row to satisfy your SQL statement or when there are no more rows to fetch (SQLCODE = 100).

The *action* of the WHENEVER statement is one of these two values:

CONTINUE
Specifies the next sequential statement of the source program.

GOTO or GO TO *host-label*
Specifies the statement identified by *host-label*. For *host-label*, substitute a single token, preceded by an optional colon. The form of the token depends on the host language. In COBOL, for example, it can be *section-name* or an unqualified *paragraph-name*.

The WHENEVER statement must precede the first SQL statement it is to affect. However, if your program checks SQLCODE directly, you must check SQLCODE after each SQL statement.

Related concepts

REXX applications that issue SQL statements
You can code SQL statements in a REXX programs wherever you can use REXX commands.

Related reference

WHENEVER (Db2 SQL)

Checking the execution of SQL statements by using the GET DIAGNOSTICS statement

One way to check whether an SQL statement executed successfully is to ask Db2 to return the diagnostic information about the last executed SQL statement.

Procedure

You can use the GET DIAGNOSTICS statement to return diagnostic information about the last SQL statement that was executed.

You can request individual items of diagnostic information from the following groups of items:

- Statement items, which contain information about the SQL statement as a whole
- Condition items, which contain information about each error or warning that occurred during the execution of the SQL statement
- Connection items, which contain information about the SQL statement if it was a CONNECT statement

In addition to requesting individual items, you can request that GET DIAGNOSTICS return all diagnostic items that are set during the execution of the last SQL statement as a single string.

In SQL procedures, you can also retrieve diagnostic information by using handlers. Handlers tell the procedure what to do if a particular error occurs.

Use the GET DIAGNOSTICS statement to handle multiple SQL errors that might result from the execution of a single SQL statement. First, check SQLSTATE (or SQLCODE) to determine whether diagnostic information should be retrieved by using GET DIAGNOSTICS. This method is especially useful for diagnosing problems that result from a multiple-row INSERT that is specified as NOT ATOMIC CONTINUE ON SQLEXCEPTIONand multiple row MERGE statements.

Even if you use only the GET DIAGNOSTICS statement in your application program to check for conditions, you must either include the instructions required to use the SQLCA or you must declare SQLSTATE (or SQLCODE) separately in your program.

When you use the GET DIAGNOSTICS statement, you assign the requested diagnostic information to host variables. Declare each target host variable with a data type that is compatible with the data type of the requested item.

To retrieve condition information, you must first retrieve the number of condition items (that is, the number of errors and warnings that Db2 detected during the execution of the last SQL statement). The number of condition items is at least one. If the last SQL statement returned SQLSTATE '00000' (or SQLCODE 0), the number of condition items is one.

Example: Using GET DIAGNOSTICS with multiple-row INSERT

You want to display diagnostic information for each condition that might occur during the execution of a multiple-row INSERT statement in your application program. You specify the INSERT statement as NOT ATOMIC CONTINUE ON SQLEXCEPTION, which means that execution continues regardless of the failure of any single-row insertion. Db2 does not insert the row that was processed at the time of the error.

In the following example, the first GET DIAGNOSTICS statement returns the number of rows inserted and the number of conditions returned. The second GET DIAGNOSTICS statement returns the following items for each condition: SQLCODE, SQLSTATE, and the number of the row (in the rowset that was being inserted) for which the condition occurred.

```
EXEC SQL BEGIN DECLARE SECTION;
  long row_count, num_condns, i;
  long ret_sqlcode, row_num;
  char ret_sqlstate[6];
  ...
EXEC SQL END DECLARE SECTION;
...
EXEC SQL
  INSERT INTO DSN8B10.ACT
```

```
    (ACTNO, ACTKWD, ACTDESC)
    VALUES (:hva1, :hva2, :hva3)
    FOR 10 ROWS
    NOT ATOMIC CONTINUE ON SQLEXCEPTION;

EXEC SQL GET DIAGNOSTICS
  :row_count = ROW_COUNT, :num_condns = NUMBER;
printf("Number of rows inserted = %d\n", row_count);

for (i=1; i<=num_condns; i++) {
  EXEC SQL GET DIAGNOSTICS CONDITION :i
    :ret_sqlcode  = DB2_RETURNED_SQLCODE,
    :ret_sqlstate = RETURNED_SQLSTATE,
    :row_num      = DB2_ROW_NUMBER;
    printf("SQLCODE = %d, SQLSTATE = %s, ROW NUMBER = %d\n",
            ret_sqlcode, ret_sqlstate, row_num);
 }
```

In the activity table, the ACTNO column is defined as SMALLINT. Suppose that you declare the host-variable array hva1 as an array with data type long, and you populate the array so that the value for the fourth element is 32768.

If you check the SQLCA values after the INSERT statement, the value of SQLCODE is equal to 0, the value of SQLSTATE is '00000', and the value of SQLERRD(3) is 9 for the number of rows that were inserted. However, the INSERT statement specified that 10 rows were to be inserted.

The GET DIAGNOSTICS statement provides you with the information that you need to correct the data for the row that was not inserted. The printed output from your program looks like this:

```
Number of rows inserted = 9
SQLCODE = -302, SQLSTATE = 22003, ROW NUMBER = 4
```

The value 32768 for the input variable is too large for the target column ACTNO. You can print the MESSAGE_TEXT condition item.

What to do next

When you use the GET DIAGNOSTICS statement, you assign the requested diagnostic information to host variables. Declare each target host variable with a data type that is compatible with the data type of the requested item.

To retrieve condition information, you must first retrieve the number of condition items (that is, the number of errors and warnings that Db2 detected during the execution of the last SQL statement). The number of condition items is at least one. If the last SQL statement returned SQLSTATE '00000' (or SQLCODE 0), the number of condition items is one.

Related concepts

Handlers in an SQL procedure
If an error occurs when an SQL procedure executes, the procedure ends unless you include statements to tell the procedure to perform some other action. These statements are called handlers.

Related reference

Data types for GET DIAGNOSTICS items
You can use the GET DIAGNOSTICS statement to request information about the statement, condition, and connection for the last SQL statement that was executed. You must declare each target host variable with a data type that is compatible with the data type of the requested item.

GET DIAGNOSTICS (Db2 SQL)

Related information

-302 (Db2 Codes)

Data types for GET DIAGNOSTICS items

You can use the GET DIAGNOSTICS statement to request information about the statement, condition, and connection for the last SQL statement that was executed. You must declare each target host variable with a data type that is compatible with the data type of the requested item.

The following tables specify the data types for the statement, condition, and connection information items that you can request by using the GET DIAGNOSTICS statement.

Table 94. Data types for GET DIAGNOSTICS items that return statement information

Item	Description	Data type
DB2_GET_DIAGNOSTICS_DIAGNOSTICS	After a GET DIAGNOSTICS statement, if any error or warning occurred, this item contains **all** of the diagnostics as a single string.	VARCHAR(32672)
DB2_LAST_ROW	After a multiple-row FETCH statement, this item contains a value of +100 if the last row in the table is in the rowset that was returned.	INTEGER
DB2_NUMBER_PARAMETER_MARKERS	After a PREPARE statement, this item contains the number of parameter markers in the prepared statement.	INTEGER
DB2_NUMBER_RESULT_SETS	After a CALL statement that invokes a stored procedure, this item contains the number of result sets that are returned by the procedure.	INTEGER
DB2_NUMBER_ROWS	After an OPEN or FETCH statement for which the size of the result table is known, this item contains the number of rows in the result table. After a PREPARE statement, this item contains the estimated number of rows in the result table for the prepared statement. For SENSITIVE DYNAMIC cursors, this item contains the approximate number of rows. Otherwise, or if the server only returns an SQLCA, the value zero is returned.	DECIMAL(31,0)
DB2_RETURN_STATUS	After a CALL statement that invokes an SQL procedure, this item contains the return status if the procedure contains a RETURN statement.	INTEGER
DB2_SQL_ATTR_CURSOR_HOLD	After an ALLOCATE or OPEN statement, this item indicates whether the cursor can be held open across multiple units of work (Y or N).	CHAR(1)
DB2_SQL_ATTR_CURSOR_ROWSET	After an ALLOCATE or OPEN statement, this item indicates whether the cursor can use rowset positioning (Y or N).	CHAR(1)

Table 94. Data types for GET DIAGNOSTICS items that return statement information (continued)

Item	Description	Data type
DB2_SQL_ATTR_CURSOR_SCROLLABLE	After an ALLOCATE or OPEN statement, this item indicates whether the cursor is scrollable (Y or N).	CHAR(1)
DB2_SQL_ATTR_CURSOR_SENSITIVITY	After an ALLOCATE or OPEN statement, this item indicates whether the cursor shows updates made by other processes (sensitivity I or S).	CHAR(1)
DB2_SQL_ATTR_CURSOR_TYPE	After an ALLOCATE or OPEN statement, this item indicates whether the cursor is forward (F), declared static (S for INSENSITIVE or SENSITIVE STATIC, or dynamic (D for SENSITIVE DYNAMIC).	CHAR(1)
DB2_SQL_NESTING_LEVEL	After a CALL statement, this item identifies the current level of nesting or recursion in effect when the GET DIAGNOSTICS statement was executed. Each level of nesting corresponds to a nested or recursive invocation of a packaged SQL function, packaged SQL procedure, or trigger. If the GET DIAGNOSTICS statement is executed outside of a level of nesting, the value zero is returned. When an application connects to another server the value is reset to zero.	INTEGER
MORE	After any SQL statement, this item indicates whether some conditions items were discarded because of insufficient storage (Y or N).	CHAR(1)
NUMBER	After any SQL statement, this item contains the number of condition items. If no warning or error occurred, or if no previous SQL statement has been executed, the number that is returned is 1.	INTEGER
ROW_COUNT	After an insert, update, delete, or fetch, this item contains the number of rows that are deleted, inserted, updated, or fetched. After PREPARE, this item contains the estimated number of result rows in the prepared statement. After TRUNCATE, it contains -1.	DECIMAL(31,0)

Table 95. Data types for GET DIAGNOSTICS items that return condition information

Item	Description	Data type
CATALOG_NAME	This item contains the server name of the table that owns a constraint that caused an error, or that caused an access rule or check violation.	VARCHAR(128)
CONDITION_NUMBER	This item contains the number of the condition.	INTEGER
CURSOR_NAME	This item contains the name of a cursor in an invalid cursor state.	VARCHAR(128)
DB2_ERROR_CODE1	This item contains an internal error code.	INTEGER
DB2_ERROR_CODE2	This item contains an internal error code.	INTEGER
DB2_ERROR_CODE3	This item contains an internal error code.	INTEGER
DB2_ERROR_CODE4	This item contains an internal error code.	INTEGER
DB2_INTERNAL_ERROR_POINTER	For some errors, this item contains a negative value that is an internal error pointer.	INTEGER
DB2_MESSAGE_ID	This item contains the message ID that corresponds to the message that is contained in the MESSAGE_TEXT diagnostic item.	CHAR(10)
DB2_MODULE_DETECTING_ERROR	After any SQL statement, this item indicates which module detected the error.	CHAR(8)
DB2_ORDINAL_TOKEN_*n*	After any SQL statement, this item contains the *n*th token, where *n* is a value from 1 to 100.	VARCHAR(515)
DB2_REASON_CODE	After any SQL statement, this item contains the reason code for errors that have a reason code token in the message text.	INTEGER
DB2_RETURNED_SQLCODE	After any SQL statement, this item contains the SQLCODE for the condition.	INTEGER
DB2_ROW_NUMBER	After any SQL statement that involves multiple rows, this item contains the row number on which Db2 detected the condition.	DECIMAL(31,0)
DB2_TOKEN_COUNT	After any SQL statement, this item contains the number of tokens available for the condition.	INTEGER
MESSAGE_TEXT	After any SQL statement, this item contains the message text associated with the SQLCODE.	VARCHAR(32672)
RETURNED_SQLSTATE	After any SQL statement, this item contains the SQLSTATE for the condition.	CHAR(5)

Table 95. Data types for GET DIAGNOSTICS items that return condition information (continued)

Item	Description	Data type
SERVER_NAME	After a CONNECT, DISCONNECT, or SET CONNECTION statement, this item contains the name of the server specified in the statement.	VARCHAR(128)

Table 96. Data types for GET DIAGNOSTICS items that return connection information

Item	Description	Data type
DB2_AUTHENTICATION_TYPE	This item contains the authentication type (S, C, D, E, or blank).	CHAR(1)
DB2_AUTHORIZATION_ID	This item contains the authorization ID that is used by the connected server.	VARCHAR(128)
DB2_CONNECTION_STATE	This item indicates whether the connection is unconnected (-1), local (0), or remote (1).	INTEGER
DB2_CONNECTION_STATUS	This item indicates whether updates can be committed for the current unit of work (1 for Yes, 2 for No).	INTEGER
DB2_ENCRYPTION_TYPE	This item contains one of the following values that indicates the level of encryption for the connection: **A** Only the authentication tokens (authid and password) are encrypted **D** All of the data for the connection is encrypted	CHAR(1)
DB2_SERVER_CLASS_NAME	After a CONNECT or SET CONNECTION statement, this item contains the Db2 server class name.	VARCHAR(128)
DB2_PRODUCT_ID	This item contains the Db2 product signature.	VARCHAR(8)

Related reference

GET DIAGNOSTICS (Db2 SQL)

Handling SQL error codes

Application programs can request more information about SQL error codes from Db2.

Procedure

Take action based on the programming language that you use.

- “Handling SQL error codes in assembler applications” on page 570
- “Handling SQL error codes in C and C++ applications” on page 617
- “Handling SQL error codes in Cobol applications” on page 683
- “Handling SQL error codes in Fortran applications” on page 692

- "Handling SQL error codes in PL/I applications" on page 702
- "Handling SQL error codes in REXX applications" on page 750

Related tasks

Displaying SQLCA fields by calling DSNTIAR
If you use the SQLCA to check whether an SQL statement executed successfully, your program needs to read the data in the appropriate SQLCA fields. One easy way to read these fields is to use the assembler subroutine DSNTIAR.

Checking the execution of SQL statements by using the GET DIAGNOSTICS statement
One way to check whether an SQL statement executed successfully is to ask Db2 to return the diagnostic information about the last executed SQL statement.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Arithmetic and conversion errors

You can track arithmetic and conversion errors by using indicator variables. An indicator variable contains a small integer value that indicates some information about the associated host variable.

Numeric or character conversion errors or arithmetic expression errors can set an indicator variable to -2. For example, division by zero and arithmetic overflow do not necessarily halt the execution of a SELECT statement. If you use indicator variables and an error occurs in the SELECT list, the statement can continue to execute and return good data for rows in which the error does not occur.

For rows in which a conversion or arithmetic expression error does occur, the indicator variable indicates that one or more selected items have no meaningful value. The indicator variable flags this error with a -2 for the affected host variable and an SQLCODE of +802 (SQLSTATE '01519') in the SQLCA.

Writing applications that enable users to create and modify tables

You can write a Db2 application that enables users to create new tables, add columns to them, increase the length of columns, rearrange the columns, and drop columns.

Procedure

To create new tables:

- Use the CREATE TABLE statement.

To add columns or increase the length of columns:

- Use the ALTER TABLE statement with the ADD COLUMN clause or the ALTER COLUMN clause.

  Added columns initially contain either the null value or a default value. Both CREATE TABLE and ALTER TABLE, like any data definition statement, are relatively expensive to execute. Also consider the effects of locks.

To drop columns:

- Use the ALTER TABLE statement with the DROP COLUMN clause.

  Dropping a column from a table is a pending-definition change unless the table space is defined with the DEFINE NO option. The column is not removed from the table until the REORG utility is run on the table space. If you are planning on dropping a column from a table in addition to making other changes to the table, make all changes that take effect immediately, prior to issuing the ALTER TABLE statement with the DROP COLUMN clause.

To rearrange columns:

- Drop the table and create the table again, with the columns you want, in the order you want.

  Consider creating a view on the table, which includes only the columns that you want, in the order that you want, as an alternative to redefining the table.

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Related reference

ALTER TABLE (Db2 SQL)
CREATE TABLE (Db2 SQL)
CREATE VIEW (Db2 SQL)

Saving SQL statements that are translated from user requests

If your program translates requests from users into SQL statements and allows users to save their requests, your program can improve performance by saving those translated statements.

About this task

A program translates requests from users into SQL statements before executing them, and users can save a request.

Procedure

Save the corresponding SQL statements in a table with a column having a data type of VARCHAR(*n*), where *n* is the maximum length of any SQL statement.

You must save the source SQL statements, not the prepared versions. That means that you must retrieve and then prepare each statement before executing the version stored in the table. In essence, your program prepares an SQL statement from a character string and executes it dynamically.

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

XML data in embedded SQL applications

Embedded SQL applications that are written in assembler language, C, C++, COBOL, or PL/I can update and retrieve data in XML columns.

In embedded SQL applications, you can:

- Store an entire XML document in an XML column using INSERT or UPDATE statements.
- Retrieve an entire XML document from an XML column using SELECT statements.
- Retrieve a sequence from a document in an XML column by using the SQL XMLQUERY function within a SELECT or FETCH statement, to retrieve the sequence into a textual XML string in the database, and then retrieve the data into an application variable.

Recommendation: Follow these guidelines when you write embedded SQL applications:

- Avoid using the XMLPARSE and XMLSERIALIZE functions.

  Let Db2 do the conversions between the external and internal XML formats implicitly.
- Use XML host variables for input and output.

  Doing so allows Db2 to process values as XML data instead of character or binary string data. If the application cannot use XML host variables, it should use binary string host variables to minimize character conversion issues.
- Avoid character conversion by using UTF-8 host variables for input and output of XML values whenever possible.

Host variable data types for XML data in embedded SQL applications

Db2 provides XML host variable types for assembler, C, C++, COBOL, and PL/I.

Those types are:

- XML AS BLOB
- XML AS CLOB
- XML AS DBCLOB
- XML AS BLOB_FILE (C, C++, or PL/I) or XML AS BLOB-FILE (COBOL)
- XML AS CLOB_FILE (C, C++, or PL/I) or XML AS CLOB-FILE (COBOL)
- XML AS DBCLOB_FILE (C, C++, or PL/I) or XML AS DBCLOB-FILE (COBOL)

The XML host variable types are compatible only with the XML column data type.

You can use BLOB, CLOB, DBCLOB, CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY, or VARBINARY host variables to update XML columns. You can convert the host variable data types to the XML type using the XMLPARSE function, or you can let the Db2 database server perform the conversion implicitly.

You can use BLOB, CLOB, DBCLOB, CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY, or VARBINARY host variables to retrieve data from XML columns. You can convert the XML data to the host variable type using the XMLSERIALIZE function, or you can let the Db2 database server perform the conversion implicitly.

The following examples show you how to declare XML host variables in each supported language. In each table, the left column contains the declaration that you code in your application program. The right column contains the declaration that Db2 generates.

Declarations of XML host variables in assembler

The following table shows assembler language declarations for some typical XML types.

Table 97. Example of assembler XML variable declarations

You declare this variable	Db2 generates this variable

`BLOB_XML SQL TYPE IS XML AS BLOB 1M`	`BLOB_XML DS 0FL4`
	`BLOB_XML_LENGTH DS FL4`
	`BLOB_XML_DATA DS CL65535`"1" on page 543
	`ORG *+(983041)`

`CLOB_XML SQL TYPE IS XML AS CLOB 40000K`	`CLOB_XML DS 0FL4`
	`CLOB_XML_LENGTH DS FL4`
	`CLOB_XML_DATA DS CL65535`"1" on page 543
	`ORG *+(40894465)`

`DBCLOB_XML SQL TYPE IS XML AS DBCLOB 4000K`	`DBCLOB_XML DS 0FL4`
	`DBCLOB_XML_LENGTH DS FL4`
	`DBCLOB_XML_DATA DS GL65534`"2" on page 543
	`ORG *+(4030466)`

`BLOB_XML_FILE SQL TYPE IS XML AS BLOB_FILE`	`BLOB_XML_FILE DS 0FL4`
	`BLOB_XML_FILE_NAME_LENGTH DS FL4`
	`BLOB_XML_FILE_DATA_LENGTH DS FL4`
	`BLOB_XML_FILE_FILE_OPTIONS DS FL4`
	`BLOB_XML_FILE_NAME DS CL255`

Table 97. Example of assembler XML variable declarations (continued)

You declare this variable	Db2 generates this variable

`CLOB_XML_FILE SQL TYPE IS XML AS CLOB_FILE`	<pre>CLOB_XML_FILE DS 0FL4
	CLOB_XML_FILE_NAME_LENGTH DS FL4
	CLOB_XML_FILE_DATA_LENGTH DS FL4
	CLOB_XML_FILE_FILE_OPTIONS DS FL4
	CLOB_XML_FILE_NAME DS CL255</pre>

`DBCLOB_XML_FILE SQL TYPE IS XML AS DBCLOB_FILE`	<pre>DBCLOB_XML_FILE DS 0FL4
	DBCLOB_XML_FILE_NAME_LENGTH DS FL4
	DBCLOB_XML_FILE_DATA_LENGTH DS FL4
	DBCLOB_XML_FILE_FILE_OPTIONS DS FL4
	DBCLOB_XML_FILE_NAME DS CL255</pre>

Notes:

1. Because assembler language allows character declarations of no more than 65535 bytes, Db2 separates the host language declarations for XML AS BLOB and XML AS CLOB host variables that are longer than 65535 bytes into two parts.
2. Because assembler language allows graphic declarations of no more than 65534 bytes, Db2 separates the host language declarations for XML AS DBCLOB host variables that are longer than 65534 bytes into two parts.

Declarations of XML host variables in C and C++

The following table shows C and C++ language declarations that are generated by the Db2 precompiler for some typical XML types. The declarations that the Db2 coprocessor generates might be different.

Table 98. Examples of C language variable declarations

You declare this variable	Db2 generates this variable

`SQL TYPE IS XML AS BLOB (1M) blob_xml;`	<pre>struct
	{ unsigned long length;
	  char data??(1048576??);
	} blob_xml;</pre>

`SQL TYPE IS XML AS CLOB(40000K) clob_xml;`	<pre>struct
	{ unsigned long length;
	  char data??(40960000??);
	} clob_xml;</pre>

`SQL TYPE IS XML AS DBCLOB (4000K) dbclob_xml;`	<pre>struct
	{ unsigned long length;
	  unsigned short data??(4096000??);
	} dbclob_xml;</pre>

`SQL TYPE IS XML AS BLOB_FILE blob_xml_file;`	<pre>struct {
	unsigned long name_length;
	unsigned long data_length;
	unsigned long file_options;
	char name??(255??);
	} blob_xml_file;</pre>

Table 98. Examples of C language variable declarations (continued)	
You declare this variable	**Db2 generates this variable**
`SQL TYPE IS XML AS CLOB_FILE clob_xml_file;`	`struct {` `unsigned long name_length;` `unsigned long data_length;` `unsigned long file_options;` `char name??(255??);` `} clob_xml_file;`
`SQL TYPE IS XML AS DBCLOB_FILE dbclob_xml_file;`	`struct {` `unsigned long name_length;` `unsigned long data_length;` `unsigned long file_options;` `char name??(255??);` `} dbclob_xml_file;`

Declarations of XML host variables in COBOL

The declarations that are generated for COBOL differ, depending on whether you use the Db2 precompiler or the Db2 coprocessor.

The following table shows COBOL declarations that the Db2 precompiler generates for some typical XML types.

Table 99. Examples of COBOL variable declarations by the Db2 precompiler

You declare this variable	Db2 precompiler generates this variable

`01  BLOB-XML USAGE IS`
`    SQL TYPE IS XML AS BLOB(1M).`
	`01  BLOB-XML.`
	`  02  BLOB-XML-LENGTH`
	`      PIC 9(9) COMP.`
	`  02  BLOB-XML-DATA.`
	`      49  FILLER PIC X(32767).`"1" on page 545
	`      49  FILLER PIC X(32767).`
	`      Repeat 30 times`
	`⋮`
	`      49  FILLER`
	`          PIC X(1048576-32*32767).`

`01  CLOB-XML USAGE IS`
`    SQL TYPE IS XML AS CLOB(40000K).`
	`01  CLOB-XML.`
	`  02  CLOB-XML-LENGTH`
	`      PIC 9(9) COMP.`
	`  02  CLOB-XML-DATA.`
	`      49  FILLER PIC X(32767).`"1" on page 545
	`      49  FILLER PIC X(32767).`
	`      Repeat 1248 times`
	`⋮`
	`      49  FILLER`
	`          PIC X(40960000-1250*32767).`

`01  DBCLOB-XML USAGE IS`
`  SQL TYPE IS XML AS DBCLOB(4000K).`
	`01  DBCLOB-XML.`
	`  02  DBCLOB-XML-LENGTH`
	`      PIC 9(9) COMP.`
	`  02  DBCLOB-XML-DATA.`
	`      49  FILLER PIC G(32767)`
	`            USAGE DISPLAY-1.`"2" on page 545
	`      49  FILLER PIC G(32767)`
	`            USAGE DISPLAY-1.`
	`      Repeat 123 times`
	`⋮`
	`      49  FILLER`
	`          PIC G(4096000-125*32767)`
	`            USAGE DISPLAY-1.`

Table 99. Examples of COBOL variable declarations by the Db2 precompiler (continued)

You declare this variable	Db2 precompiler generates this variable

`01  BLOB-XML-FILE USAGE IS SQL`
`  TYPE IS XML AS BLOB-FILE.`
	`01  BLOB-XML-FILE.`
	`  49 BLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.`
	`  49 BLOB-XML-FILE-DATA-LENGTH PIC S9(9) COMP-5.`
	`  49 BLOB-XML-FILE-FILE-OPTION PIC S9(9) COMP-5.`
	`  49 BLOB-XML-FILE-NAME PIC X(255).`

`01  CLOB-XML-FILE USAGE IS SQL`
`    TYPE IS XML AS CLOB-FILE.`
	`01  CLOB-XML-FILE.`
	`  49 CLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC.`
	`  49 CLOB-XML-FILE-DATA-LENGTH PIC S9(9) COMP-5.`
	`  49 CLOB-XML-FILE-FILE-OPTION PIC S9(9) COMP-5.`
	`  49 CLOB-XML-FILE-NAME PIC X(255).`

`01  DBCLOB-XML-FILE USAGE IS SQL`
`    TYPE IS XML AS DBCLOB-FILE.`
	`01  DBCLOB-XML-`
	`FILE.`
	`  49 DBCLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5`
	`SYNC.`
	`  49 DBCLOB-XML-FILE-DATA-LENGTH PIC S9(9)`
	`COMP-5.`
	`  49 DBCLOB-XML-FILE-FILE-OPTION PIC S9(9)`
	`COMP-5.`
	`  49 DBCLOB-XML-FILE-NAME PIC`
	`X(255).`

Notes:

1. For XML AS BLOB or XML AS CLOB host variables that are greater than 32767 bytes in length, Db2 creates multiple host language declarations of 32767 or fewer bytes.
2. For XML AS DBCLOB host variables that are greater than 32767 double-byte characters in length, Db2 creates multiple host language declarations of 32767 or fewer double-byte characters.

Declarations of XML host variables in PL/I

The declarations that are generated for PL/I differ, depending on whether you use the Db2 precompiler or the Db2 coprocessor.

The following table shows PL/I declarations that the Db2 precompiler generates for some typical XML types.

Table 100. Examples of PL/I variable declarations

You declare this variable	Db2 precompiler generates this variable

`DCL BLOB_XML`
`  SQL TYPE IS XML AS BLOB (1M);`
	`DCL`
	`   1   BLOB_XML,`
	`      2   BLOB_XML_LENGTH BIN FIXED(31),`
	`      2   BLOB_XML_DATA,`“1” on page 547
	`        3   BLOB_XML_DATA1 (32) CHAR(32767),`
	`        3   BLOB_XML_DATA2 CHAR(32);`

`DCL CLOB_XML`
`  SQL TYPE IS XML AS CLOB (40000K);`
	`DCL`
	`   1   CLOB_XML,`
	`      2   CLOB_XML_LENGTH BIN FIXED(31),`
	`      2   CLOB_XML_DATA,`“1” on page 547
	`        3   CLOB_XML_DATA1 (1250) CHAR(32767),`
	`        3   CLOB_XML_DATA2 CHAR(1250);`

Table 100. Examples of PL/I variable declarations (continued)

You declare this variable	Db2 precompiler generates this variable
`DCL DBCLOB_XML SQL TYPE IS XML AS DBCLOB (4000K);`	`DCL 1 DBCLOB_XML, 2 DBCLOB_XML_LENGTH BIN FIXED(31), 2 DBCLOB_XML_DATA,` "2" on page 547 `3 DBCLOB_XML_DATA1 (250 ) GRAPHIC(16383), 3 DBCLOB_XML_DATA2 GRAPHIC(250);`
`DCL BLOB_XML_FILE SQL TYPE IS XML AS BLOB_FILE;`	`DCL 1 BLOB_XML_FILE, 2 BLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 BLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 BLOB_XML_FILE_FILE_OPTIONS BIN FIXED(31), 2 BLOB_XML_FILE_NAME CHAR(255);`
`DCL CLOB_XML_FILE SQL TYPE IS XML AS CLOB_FILE;`	`DCL 1 CLOB_XML_FILE, 2 CLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 CLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 CLOB_XML_FILE_FILE_OPTIONS BIN FIXED(31), 2 CLOB_XML_FILE_NAME CHAR(255);`
`DCL DBCLOB_XML_FILE SQL TYPE IS XML AS DBCLOB_FILE;`	`DCL 1 DBCLOB_XML_FILE, 2 DBCLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 DBCLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 DBCLOB_XML_FILE_FILE_OPTIONS BIN FIXED(31), 2 DBCLOB_XML_FILE_NAME CHAR(255);`

Table 100. Examples of PL/I variable declarations (continued)	
You declare this variable	**Db2 precompiler generates this variable**

Notes:

1. For XML AS BLOB or XML AS CLOB host variables that are greater than 32767 bytes in length, Db2 creates host language declarations in the following way:
   - If the length of the XML is greater than 32767 bytes and evenly divisible by 32767, Db2 creates an array of 32767-byte strings. The dimension of the array is *length*/32767.
   - If the length of the XML is greater than 32767 bytes but not evenly divisible by 32767, Db2 creates two declarations: The first is an array of 32767 byte strings, where the dimension of the array, *n*, is *length*/32767. The second is a character string of length *length*-*n**32767.
2. For XML AS DBCLOB host variables that are greater than 16383 double-byte characters in length, Db2 creates host language declarations in the following way:
   - If the length of the XML is greater than 16383 characters and evenly divisible by 16383, Db2 creates an array of 16383-character strings. The dimension of the array is *length*/16383.
   - If the length of the XML is greater than 16383 characters but not evenly divisible by 16383, Db2 creates two declarations: The first is an array of 16383 byte strings, where the dimension of the array, *m*, is *length*/16383. The second is a character string of length *length*-*m**16383.

Related concepts
Insertion of rows with XML column values (Db2 Programming for XML)
Retrieving XML data (Db2 Programming for XML)
Updates of XML columns (Db2 Programming for XML)

XML column updates in embedded SQL applications

When you update or insert data into XML columns of a Db2 table, the input data must be in the textual XML format.

The encoding of XML data can be derived from the data itself, which is known as *internally encoded* data, or from external sources, which is known as *externally encoded* data. XML data that is sent to the database server as binary data is treated as internally encoded data. XML data that is sent to the database server as character data is treated as externally encoded data.

Externally encoded data can have internal encoding. That is, the data might be sent to the database server as character data, but the data contains encoding information. Db2 does not enforce consistency of the internal and external encoding. When the internal and external encoding information differs, the external encoding takes precedence. However, if there is a difference between the external and internal encoding, intervening character conversion might have occurred on the data, and there might be data loss.

Character data in XML columns is stored in UTF-8 encoding. The database server handles conversion of the data from its internal or external encoding to UTF-8.

The following examples demonstrate how to update XML columns in assembler, C, COBOL, and PL/I applications. The examples use a table named MYCUSTOMER, which is a copy of the sample CUSTOMER table.

Example

The following example shows an assembler program that inserts data from XML AS BLOB, XML AS CLOB, and CLOB host variables into an XML column. The XML AS BLOB data is inserted as binary data, so the database server honors the internal encoding. The XML AS CLOB and CLOB data is inserted as character data, so the database server honors the external encoding.

```
**********************************************************************
* UPDATE AN XML COLUMN WITH DATA IN AN XML AS CLOB HOST VARIABLE     *
**********************************************************************
```

```
         EXEC SQL                                                      +
               UPDATE MYCUSTOMER                                       +
               SET INFO = :XMLBUF                                      +
               WHERE CID = 1000
**********************************************************************
* UPDATE AN XML COLUMN WITH DATA IN AN XML AS BLOB HOST VARIABLE     *
**********************************************************************
         EXEC SQL                                                      +
               UPDATE MYCUSTOMER                                       +
               SET INFO = :XMLBLOB                                     +
               WHERE CID = 1000
**********************************************************************
* UPDATE AN XML COLUMN WITH DATA IN A CLOB HOST VARIABLE. USE        *
* THE XMLPARSE FUNCTION TO CONVERT THE DATA TO THE XML TYPE.         *
**********************************************************************
         EXEC SQL                                                      +
               UPDATE MYCUSTOMER                                       +
               SET INFO = XMLPARSE(DOCUMENT :CLOBBUF)                  +
               WHERE CID = 1000
…
          LTORG
******************************
* HOST VARIABLE DECLARATIONS *
******************************
XMLBUF   SQL TYPE IS XML AS CLOB 10K
XMLBLOB  SQL TYPE IS XML AS BLOB 10K
CLOBBUF  SQL TYPE IS CLOB 10K
```

Example

The following example shows a C language program that inserts data from XML AS BLOB, XML AS CLOB, and CLOB host variables into an XML column. The XML AS BLOB data is inserted as binary data, so the database server honors the internal encoding. The XML AS CLOB and CLOB data is inserted as character data, so the database server honors the external encoding.

```
/******************************/
/* Host variable declarations */
/******************************/
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;
/******************************************************************/
/* Update an XML column with data in an XML AS CLOB host variable */
/******************************************************************/
EXEC SQL UPDATE MYCUSTOMER SET INFO = :xmlBuf where CID = 1000;
/******************************************************************/
/* Update an XML column with data in an XML AS BLOB host variable */
/******************************************************************/
 EXEC SQL UPDATE MYCUSTOMER SET INFO = :xmlblob where CID = 1000;
/******************************************************************/
/* Update an XML column with data in a CLOB host variable. Use    */
/* the XMLPARSE function to convert the data to the XML type.     */
/******************************************************************/
EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :clobBuf) where CID = 1000;
```

Example

The following example shows a COBOL program that inserts data from XML AS BLOB, XML AS CLOB, and CLOB host variables into an XML column. The XML AS BLOB data is inserted as binary data, so the database server honors the internal encoding. The XML AS CLOB and CLOB data is inserted as character data, so the database server honors the external encoding.

```
******************************
* Host variable declarations *
******************************
 01 XMLBUF USAGE IS SQL TYPE IS XML as CLOB(10K).
 01 XMLBLOB  USAGE IS SQL TYPE IS XML AS BLOB(10K).
 01 CLOBBUF USAGE IS SQL TYPE IS CLOB(10K).
*******************************************************************
* Update an XML column with data in an XML AS CLOB host variable  *
*******************************************************************
 EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBUF where CID = 1000.
*******************************************************************
* Update an XML column with data in an XML AS BLOB host variable  *
*******************************************************************
 EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBLOB where CID = 1000.
```

```
******************************************************************
* Update an XML column with data in a CLOB host variable. Use     *
* the XMLPARSE function to convert the data to the XML type.      *
******************************************************************
 EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :CLOBBUF) where CID = 1000.
```

Example

The following example shows a PL/I program that inserts data from XML AS BLOB, XML AS CLOB, and CLOB host variables into an XML column. The XML AS BLOB data is inserted as binary data, so the database server honors the internal encoding. The XML AS CLOB and CLOB data is inserted as character data, so the database server honors the external encoding.

```
/*******************************/
/* Host variable declarations */
/*******************************/
 DCL
  XMLBUF SQL TYPE IS XML AS CLOB(10K),
  XMLBLOB SQL TYPE IS XML AS BLOB(10K),
  CLOBBUF SQL TYPE IS CLOB(10K);
/*******************************************************************/
/* Update an XML column with data in an XML AS CLOB host variable  */
/*******************************************************************/
 EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBUF where CID = 1000;
/*******************************************************************/
/* Update an XML column with data in an XML AS BLOB host variable  */
/*******************************************************************/
 EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBLOB where CID = 1000;
/*******************************************************************/
/* Update an XML column with data in a CLOB host variable. Use     */
/* the XMLPARSE function to convert the data to the XML type.      */
/*******************************************************************/
 EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :CLOBBUF) where CID = 1000;
```

Related concepts

Insertion of rows with XML column values (Db2 Programming for XML)

Updates of XML columns (Db2 Programming for XML)

XML data retrieval in embedded SQL applications

In an embedded SQL application, if you retrieve the data into a character host variable, Db2 converts the data from the UTF-8 encoding scheme to the application encoding scheme. If you retrieve the data into binary host variable, Db2 does not convert the data to another encoding scheme.

The output data is in the textual XML format.

Db2 might add an XML encoding specification to the retrieved data, depending on whether you call the XMLSERIALIZE function when you retrieve the data. If you do not call the XMLSERIALIZE function, Db2 adds the correct XML encoding specification to the retrieved data. If you call the XMLSERIALIZE function, Db2 adds an internal XML encoding declaration for UTF-8 encoding if you specify INCLUDING XMLDECLARATION in the function call. When you use INCLUDING XMLDECLARATION, you need to ensure that the retrieved data is not converted from UTF-8 encoding to another encoding.

The following examples demonstrate how to retrieve data from XML columns in assembler, C, COBOL, and PL/I applications. The examples use a table named MYCUSTOMER, which is a copy of the sample CUSTOMER table.

Example: The following example shows an assembler program that retrieves data from an XML column into XML AS BLOB, XML AS CLOB, and CLOB host variables. The data that is retrieved into an XML AS BLOB host variable is retrieved as binary data, so the database server generates an XML declaration with UTF-8 encoding. The data that is retrieved into an XML AS CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration that is consistent with the external encoding. The data that is retrieved into a CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration. That declaration might not be consistent with the external encoding.

```
**********************************************************************
* RETRIEVE XML COLUMN DATA INTO AN XML AS CLOB HOST VARIABLE         *
**********************************************************************
```

```
         EXEC SQL                                                        +
               SELECT INFO                                               +
               INTO :XMLBUF                                              +
               FROM MYCUSTOMER                                           +
               WHERE CID = 1000
**********************************************************************
* RETRIEVE XML COLUMN DATA INTO AN XML AS BLOB HOST VARIABLE         *
**********************************************************************
         EXEC SQL                                                        +
               SELECT INFO                                               +
               INTO :XMLBLOB                                             +
               FROM MYCUSTOMER                                           +
               WHERE CID = 1000
**********************************************************************
* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.        *
* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE             *
* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML             *
* TYPE TO THE CLOB TYPE.                                             *
**********************************************************************
         EXEC SQL                                                        +
               SELECT XMLSERIALIZE(INFO AS CLOB(10K))                    +
               INTO :CLOBBUF                                             +
               FROM MYCUSTOMER                                           +
               WHERE CID = 1000
…
          LTORG
******************************
* HOST VARIABLE DECLARATIONS *
******************************
XMLBUF    SQL TYPE IS XML AS CLOB 10K
XMLBLOB   SQL TYPE IS XML AS BLOB 10K
CLOBBUF   SQL TYPE IS CLOB 10K
```

Example: The following example shows a C language program that retrieves data from an XML column into XML AS BLOB, XML AS CLOB, and CLOB host variables. The data that is retrieved into an XML AS BLOB host variable is retrieved as binary data, so the database server generates an XML declaration with UTF-8 encoding. The data that is retrieved into an XML AS CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration that is consistent with the external encoding. The data that is retrieved into a CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration. That declaration might not be consistent with the external encoding.

```
/******************************/
/* Host variable declarations */
/******************************/
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
SQL TYPE IS XML AS BLOB( 10K ) xmlBlob;
SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;
/**********************************************************************/
/* Retrieve data from an XML column into an XML AS CLOB host variable */
/**********************************************************************/
EXEC SQL SELECT INFO   INTO :xmlBuf from myTable where CID = 1000;
/**********************************************************************/
/* Retrieve data from an XML column into an XML AS BLOB host variable */
/**********************************************************************/
EXEC SQL SELECT INFO   INTO :xmlBlob from myTable where CID = 1000;
/**********************************************************************/
/* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.        */
/* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE             */
/* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML             */
/* TYPE TO THE CLOB TYPE.                                             */
/**********************************************************************/
EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
 INTO :clobBuf from myTable where CID = 1000;
```

Example: The following example shows a COBOL program that retrieves data from an XML column into XML AS BLOB, XML AS CLOB, and CLOB host variables. The data that is retrieved into an XML AS BLOB host variable is retrieved as binary data, so the database server generates an XML declaration with UTF-8 encoding. The data that is retrieved into an XML AS CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration that is consistent with the external encoding. The data that is retrieved into a CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration. That declaration might not be consistent with the external encoding.

```
******************************
* Host variable declarations *
******************************
 01 XMLBUF  USAGE IS SQL TYPE IS XML AS CLOB(10K).
 01 XMLBLOB USAGE IS SQL TYPE IS XML AS BLOB(10K).
 01 CLOBBUF USAGE IS SQL TYPE IS CLOB(10K).
**********************************************************************
* Retrieve data from an XML column into an XML AS CLOB host variable *
**********************************************************************
 EXEC SQL SELECT INFO
   INTO :XMLBUF
   FROM MYTABLE
   WHERE CID = 1000
 END-EXEC.
**********************************************************************
* Retrieve data from an XML column into an XML AS BLOB host variable *
**********************************************************************
 EXEC SQL SELECT INFO
   INTO :XMLBLOB
   FROM MYTABLE
   WHERE CID = 1000
 END-EXEC.
**********************************************************************
* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.        *
* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE             *
* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML             *
* TYPE TO THE CLOB TYPE.                                             *
**********************************************************************
 EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
  INTO :CLOBBUF
  FROM MYTABLE
  WHERE CID = 1000
 END-EXEC.
```

Example: The following example shows a PL/I program that retrieves data from an XML column into XML AS BLOB, XML AS CLOB, and CLOB host variables. The data that is retrieved into an XML AS BLOB host variable is retrieved as binary data, so the database server generates an XML declaration with UTF-8 encoding. The data that is retrieved into an XML AS CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration that is consistent with the external encoding. The data that is retrieved into a CLOB host variable is retrieved as character data, so the database server generates an XML declaration with an internal encoding declaration. That declaration might not be consistent with the external encoding.

```
/******************************/
/* Host variable declarations */
/******************************/
 DCL
  XMLBUF SQL TYPE IS XML AS CLOB(10K),
  XMLBLOB SQL TYPE IS XML AS BLOB(10K),
  CLOBBUF SQL TYPE IS CLOB(10K);
/***********************************************************************/
/* Retrieve data from an XML column into an XML AS CLOB host variable */
/***********************************************************************/
 EXEC SQL SELECT INFO   INTO :XMLBUF FROM MYTABLE WHERE CID = 1000;
/***********************************************************************/
/* Retrieve data from an XML column into an XML AS BLOB host variable */
/***********************************************************************/
 EXEC SQL SELECT INFO   INTO :XMLBLOB FROM MYTABLE WHERE CID = 1000;
/***********************************************************************/
/* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.        */
/* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE             */
/* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML             */
/* TYPE TO THE CLOB TYPE.                                             */
/***********************************************************************/
 EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
  INTO :CLOBBUF FROM MYTABLE WHERE CID = 1000;
```

Retrieving XML data (Db2 Programming for XML)

Example programs that call stored procedures

Examples can be used as models when you write applications that call stored procedures. In addition, *prefix*.SDSNSAMP contains sample jobs DSNTEJ6P and DSNTEJ6S and programs DSN8EP1 and DSN8EP2, which you can run.

Related concepts

Sample applications supplied with Db2 for z/OS

Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

Assembler applications that issue SQL statements

You can code SQL statements in assembler programs wherever you can use executable statements.

Each SQL statement in an assembler program must begin with EXEC SQL. The EXEC and SQL keywords must appear on one line, but the remainder of the statement can appear on subsequent lines.

You might code an UPDATE statement in an assembler program as follows:

```
EXEC SQL UPDATE DSN8B10.DEPT                          X
            SET MGRNO = :MGRNUM                       X
            WHERE DEPTNO = :INTDEPT
```

Comments

You cannot include assembler comments in SQL statements. However, you can include SQL comments in any embedded SQL statement. For more information, see SQL comments (Db2 SQL).

Continuation for SQL statements

The line continuation rules for SQL statements are the same as those for assembler statements, except that you must specify EXEC SQL within one line. Any part of the statement that does not fit on one line can appear on subsequent lines, beginning at the continuation margin (column 16, the default). Every line of the statement, except the last, must have a continuation character (a non-blank character) immediately after the right margin in column 72.

Delimiters for SQL statements

Delimit an SQL statement in your assembler program with the beginning keyword EXEC SQL and an end of line or end of last continued line.

Declaring tables and views

Your assembler program should include a DECLARE statement to describe each table and view the program accesses.

Including code

To include SQL statements or assembler host variable declaration statements from a member of a partitioned data set, place the following SQL statement in the source code where you want to include the statements:

```
EXEC SQL INCLUDE member-name
```

You cannot nest SQL INCLUDE statements.

Margins

Use the precompiler option MARGINS to set a left margin, a right margin, and a continuation margin. The default values for these margins are columns 1, 71, and 16, respectively. If EXEC SQL starts before the specified left margin, the Db2 precompiler does not recognize the SQL statement. If you use the default margins, you can place an SQL statement anywhere between columns 2 and 71.

Multiple-row FETCH statements

You can use only the FETCH ... USING DESCRIPTOR form of the multiple-row FETCH statement in an assembler program. The Db2 precompiler does not recognize declarations of host-variable arrays for an assembler program.

Names

You can use any valid assembler name for a host variable. However, do not use external entry names or access plan names that begin with 'DSN' or host variable names that begin with 'SQL'. These names are reserved for Db2.

The first character of a host variable that is used in embedded SQL cannot be an underscore. However, you can use an underscore as the first character in a symbol that is not used in embedded SQL.

Statement labels

You can prefix an SQL statement with a label. The first line of an SQL statement can use a label beginning in the left margin (column 1). If you do not use a label, leave column 1 blank.

WHENEVER statement

The target for the GOTO clause in an SQL WHENEVER statement must be a label in the assembler source code and must be within the scope of the SQL statements that WHENEVER affects.

Special assembler considerations

The following considerations apply to programs written in assembler:

- To allow for reentrant programs, the precompiler puts all the variables and structures it generates within a DSECT called SQLDSECT, and it generates an assembler symbol called SQLDLEN. SQLDLEN contains the length of the DSECT. Your program must allocate an area of the size indicated by SQLDLEN, initialize it, and provide addressability to it as the DSECT SQLDSECT. The precompiler does not generate code to allocate the storage for SQLDSECT; the application program must allocate the storage.

  CICS: An example of code to support reentrant programs, running under CICS, follows:

```
DFHEISTG DSECT
         DFHEISTG
         EXEC SQL INCLUDE SQLCA
*
         DS    0F
SQDWSREG EQU   R7
SQDWSTOR DS    (SQLDLEN)C  RESERVE STORAGE TO BE USED FOR SQLDSECT

⋮

XXPROGRM DFHEIENT CODEREG=R12,EIBREG=R11,DATAREG=R13
*
*
*  SQL WORKING STORAGE
         LA    SQDWSREG,SQDWSTOR     GET ADDRESS OF SQLDSECT
         USING SQLDSECT,SQDWSREG     AND TELL ASSEMBLER ABOUT IT
*
```

  In this example, the actual storage allocation is done by the DFHEIENT macro.

  TSO: The sample program in *prefix*.SDSNSAMP(DSNTIAD) contains an example of how to acquire storage for the SQLDSECT in a program that runs in a TSO environment. The following example code contains pieces from *prefix*.SDSNSAMP(DSNTIAD) with explanations in the comments.

```
DSNTIAD  CSECT                  CONTROL SECTION NAME
         SAVE  (14,12)          ANY SAVE SEQUENCE
         LR    R12,R15          CODE ADDRESSABILITY
         USING DSNTIAD,R12      TELL THE ASSEMBLER
         LR    R7,R1            SAVE THE PARM POINTER
*
* Allocate storage of size PRGSIZ1+SQLDSIZ, where:
* - PRGSIZ1 is the size of the DSNTIAD program area
* - SQLDSIZ is the size of the SQLDSECT, and declared
*   when the DB2 precompiler includes the SQLDSECT
*
         L     R6,PRGSIZ1       GET SPACE FOR USER PROGRAM
         A     R6,SQLDSIZ       GET SPACE FOR SQLDSECT
         GETMAIN R,LV=(6)       GET STORAGE FOR PROGRAM VARIABLES
         LR    R10,R1           POINT TO IT
*
* Initialize the storage
*
         LR    R2,R10           POINT TO THE FIELD
         LR    R3,R6            GET ITS LENGTH
```

```
         SR    R4,R4          CLEAR THE INPUT ADDRESS
         SR    R5,R5          CLEAR THE INPUT LENGTH
         MVCL  R2,R4          CLEAR OUT THE FIELD
*
* Map the storage for DSNTIAD program area
*
         ST    R13,FOUR(R10)        CHAIN THE SAVEAREA PTRS
         ST    R10,EIGHT(R13)       CHAIN SAVEAREA FORWARD
         LR    R13,R10              POINT TO THE SAVEAREA
         USING PRGAREA1,R13         SET ADDRESSABILITY
*
* Map the storage for the SQLDSECT
*
         LR    R9,R13               POINT TO THE PROGAREA
         A     R9,PRGSIZ1           THEN PAST TO THE SQLDSECT
         USING SQLDSECT,R9          SET ADDRESSABILITY
...
         LTORG
**********************************************************************
*                                                                    *
*    DECLARE VARIABLES, WORK AREAS                                   *
*                                                                    *
**********************************************************************
PRGAREA1 DSECT                      WORKING STORAGE FOR THE PROGRAM
...
         DS    0D
PRGSIZE1 EQU   *-PRGAREA1           DYNAMIC WORKAREA SIZE
...
DSNTIAD  CSECT                      RETURN TO CSECT FOR CONSTANT
PRGSIZ1  DC    A(PRGSIZE1)          SIZE OF PROGRAM WORKING STORAGE
CA       DSECT
         EXEC SQL INCLUDE SQLCA
...
```

- Db2 does not process set symbols in SQL statements.
- Generated code can include more than two continuations per comment.
- Generated code uses literal constants (for example, =F'-84'), so an LTORG statement might be necessary.
- Generated code uses registers 0, 1, 14, and 15. Register 13 points to a save area that the called program uses. Register 15 does not contain a return code after a call that is generated by an SQL statement.

  CICS: A CICS application program uses the DFHEIENT macro to generate the entry point code. When using this macro, consider the following:

  – If you use the default DATAREG in the DFHEIENT macro, register 13 points to the save area.

  – If you use any other DATAREG in the DFHEIENT macro, you must provide addressability to a save area.

    For example, to use SAVED, you can code instructions to save, load, and restore register 13 around each SQL statement as in the following example.

    ```
    ST    13,SAVER13       SAVE REGISTER 13
    LA    13,SAVED         POINT TO SAVE AREA
    EXEC  SQL . . .
    L     13,SAVER13       RESTORE REGISTER 13
    ```

- If you have an addressability error in precompiler-generated code because of input or output host variables in an SQL statement, check to make sure that you have enough base registers.
- Do not put CICS translator options in the assembly source code. Instead, pass the options to the translator by using the PARM field.

Handling SQL error codes

Assembler applications can request more information about SQL errors from Db2. For more information, see "Handling SQL error codes in assembler applications" on page 570.

Related tasks

Overview of programming applications that access Db2 for z/OS data

Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Assembler programming examples

You can write Db2 programs in assembler. These programs can access a local or remote Db2 subsystem and can execute static or dynamic SQL statements. This information contains several such programming examples.

To prepare and run these applications, use the JCL in *prefix*.SDSNSAMP as a model for your JCL.

Related reference

Application languages and environments for the sample applications
The sample applications demonstrate how to run Db2 applications in the TSO, IMS, or CICS environments.

Defining the SQL communications area, SQLSTATE, and SQLCODE in assembler

Assembler programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

About this task

If you specify the SQL processing option STDSQL(YES), do not define an SQLCA. If you do, Db2 ignores your SQLCA, and your SQLCA definition causes compile-time errors. If you specify the SQL processing option STDSQL(NO), include an SQLCA explicitly.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Procedure

Choose one of the following actions:

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>To define the SQL communications area:</td><td>
a. Code the SQLCA directly in the program or use the following SQL INCLUDE statement to request a standard SQLCA declaration:
<pre>EXEC SQL INCLUDE SQLCA</pre>
If your program is reentrant, you must include the SQLCA within a unique data area that is acquired for your task (a DSECT). For example, at the beginning of your program, specify the following code:
<pre>PROGAREA DSECT
         EXEC SQL INCLUDE SQLCA</pre>
As an alternative, you can create a separate storage area for the SQLCA and provide addressability to that area.

Db2 sets the SQLCODE and SQLSTATE values in the SQLCA after each SQL statement executes. Your application should check these values to determine whether the last SQL statement was successful.
</td></tr>
</table>

Option	Description
To declare SQLCODE and SQLSTATE host variables:	a. Declare the SQLCODE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as a fullword integer. b. Declare the SQLSTATE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as a character string of length 5 (CL5). **Restriction:** Do not declare an SQLSTATE variable as an element of a structure. **Requirement:** After you declare the SQLCODE and SQLSTATE variables, ensure that all SQL statements in the program are within the scope of the declaration of these variables.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas (SQLDA) in assembler

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Code the SQLDA directly in the program, or use the following SQL INCLUDE statement to request a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA
```

Restriction: You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.

Example

You can use host-variable arrays for certain multi-row operations in other host languages, such C, C++, COBOL, and PL/I. but the Db2 precompiler does not recognize declarations of host-variable arrays for assembler. However, you can SQLDA declarations to achieve similar results in assembler programs, as shown in the following examples:

- Assembler support for multiple-row FETCH is limited to the FETCH statement with the INTO DESCRIPTOR clause. For example:

```
EXEC SQL FETCH NEXT ROWSET FROM C1 FOR 10 ROWS                      X
         INTO DESCRIPTOR :SQLDA
```

- Assembler support for multiple-row INSERT is limited to the following cases:
  - Static multiple-row INSERT statement with scalar values (scalar host variables or scalar expressions) in the VALUES clause. For example:

    ```
    EXEC SQL INSERT INTO T1 VALUES (1, CURRENT DATE, 'TEST')               X
             FOR 10 ROWS
    ```

  - Dynamic multiple-row INSERT executed with the USING DESCRIPTOR clause on the EXECUTE statement. For example:

    ```
    ATR       DS    CL20                         ATTRIBUTES FOR PREPARE
    S1        DS    H,CL30                       VARCHAR STATEMENT STRING
              MVC   ATR(20),=C'FOR MULTIPLE ROWS '
              MVC   S1(2),=H'25'
              MVC   S1+2(30),=C'INSERT INTO T1 VALUES (?) '
              EXEC  SQL PREPARE STMT ATTRIBUTES :ATR FROM :S1
              EXEC  SQL EXECUTE STMT USING DESCRIPTOR :SQLDA FOR 10 ROWS
    where the descriptor is set up correctly in advance according to the
    specifications for dynamic execution of a multiple-row INSERT statement
    with a descriptor
    ```

  - Assembler does not support multiple-row MERGE. You cannot specify MERGE statements that reference host-variable arrays.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Declaring host variables and indicator variables in assembler

You can use host variables, host-variable arrays, and host structures in SQL statements in your program to pass data between Db2 and your application.

Procedure

To declare host variables, host-variable arrays, and host structures:

1. Declare the variables according to the following rules and guidelines:
   - You can declare host variables in normal assembler style (DC or DS), depending on the data type and the limitations on that data type. You can specify a value on DC or DS declarations (for example, `DC H'5'`). The Db2 precompiler examines only packed decimal declarations.
   - If you specify the ONEPASS SQL processing option, you must explicitly declare each host variable and each host-variable array before using them in an SQL statement. If you specify the TWOPASS precompiler option, you must declare each host variable before using it in the DECLARE CURSOR statement.
   - If you specify the STDSQL(YES) SQL processing option, you must precede the host language statements that define the host variables and host-variable arrays with the BEGIN DECLARE SECTION statement and follow the host language statements with the END DECLARE SECTION statement. Otherwise, these statements are optional.
   - Ensure that any SQL statement that uses a host variable or host-variable array is within the scope of the statement that declares that variable or array.
   - If you are using the Db2 precompiler, ensure that the names of host variables and host-variable arrays are unique within the program, even if the variables and variable arrays are in different blocks,

classes, procedures, functions, or subroutines. You can qualify the names with a structure name to make them unique.

2. Optional: Define any associated indicator variables, arrays, and structures.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Host variables in assembler

In assembler programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Restrictions:

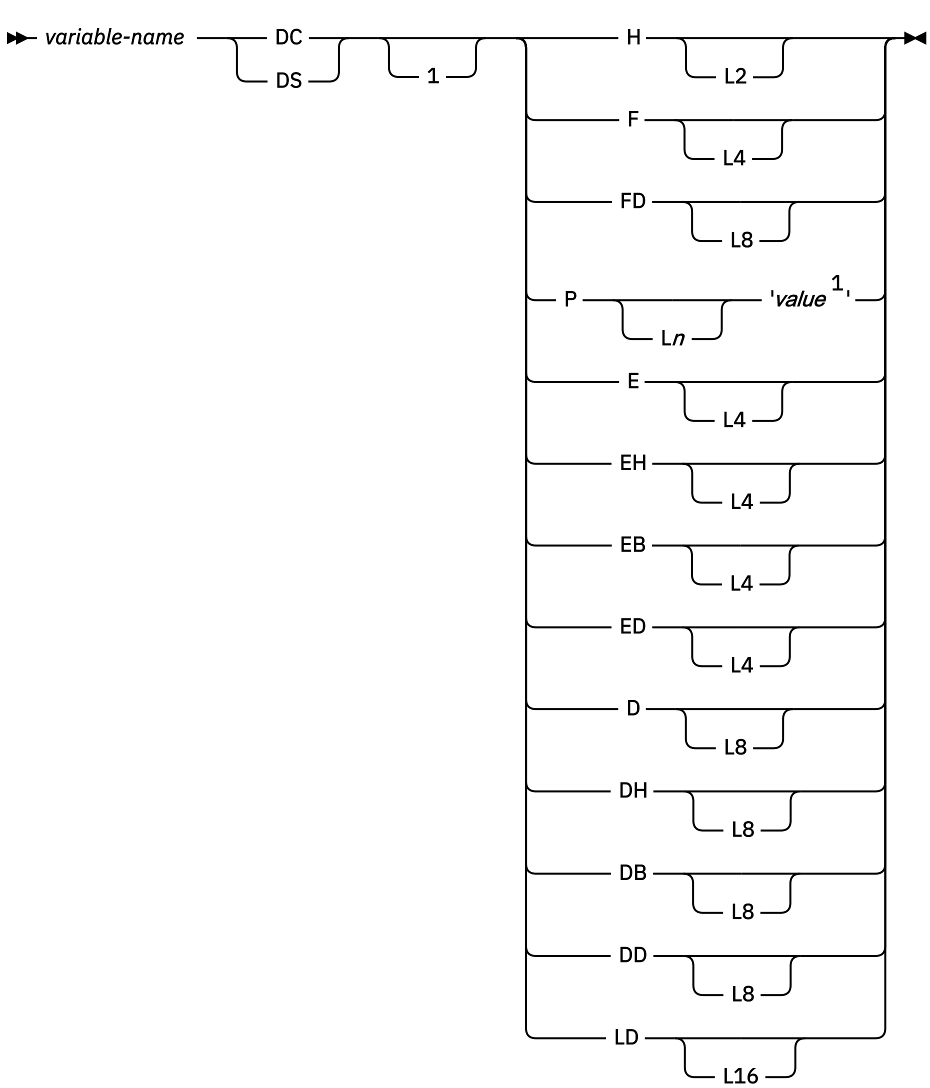
- Only some of the valid assembler declarations are valid host variable declarations. If the declaration for a host variable is not valid, any SQL statement that references the variable might result in the message UNDECLARED HOST VARIABLE.
- The locator data types are assembler language data types and SQL data types. You cannot use locators as column types.

Recommendations:

- Be careful of overflow. For example, suppose that you retrieve an INTEGER column value into a DS H host variable, and the column value is larger than 32767. You get an overflow warning or an error, depending on whether you provide an indicator variable.
- Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a host variable that is declared as DS CL70, the rightmost ten characters of the retrieved string are truncated. If you retrieve a floating-point or decimal column value into a host variable declared as DS F, any fractional part of the value is removed.

Numeric host variables

The following diagram shows the syntax for declaring numeric host variables.



Notes:

[1] *value* is a numeric value that specifies the scale of the packed decimal variable. If *value* does not include a decimal point, the scale is 0.

For floating-point data types (E, EH, EB, D, DH, and DB), use the FLOAT SQL processing option to specify whether the host variable is in IEEE binary floating-point or z/Architecture® hexadecimal floating-point format. If you specify FLOAT(S390), you need to define your floating-point host variables as E, EH, D, or DH. If you specify FLOAT(IEEE), you need to define your floating-point host variables as EB or DB. Db2 does not check if the host variable declarations or format of the host variable contents match the format that you specified with the FLOAT SQL processing option. Therefore, you need to ensure that your floating-point host variable types and contents match the format that you specified with the FLOAT SQL processing option. Db2 converts all floating-point input data to z/Architecture hexadecimal floating-point format before storing it.

Restriction: The FLOAT SQL processing options do not apply to the decimal floating-point host variable types ED, DD, or LD.

For the decimal floating-point host variable types ED, DD, and LD, you can specify the following special values: MIN, MAX, NAN, SNAN, and INFINITY.

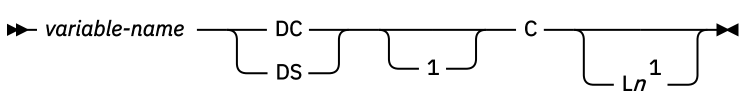
Character host variables

You can specify the following forms of character host variables:

- Fixed-length strings
- Varying-length strings
- CLOBs

The following diagrams show the syntax for forms other than CLOBs.

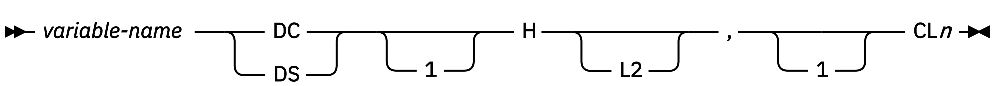
The following diagram shows the syntax for declaring fixed-length character strings.



Notes:

[1] If you declare a character string host variable without a length (for example, `DC C 'ABCD'`) Db2 interprets the length as 1. To get the correct length, specify a length attribute (for example, `DC CL 4 'ABCD'`).

The following diagram shows the syntax for declaring varying-length character strings.



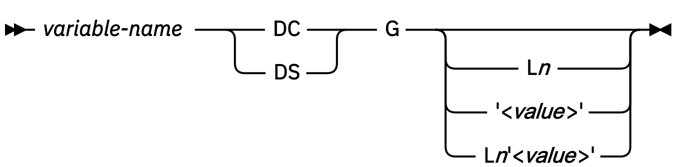
Graphic host variables

You can specify the following forms of graphic host variables:

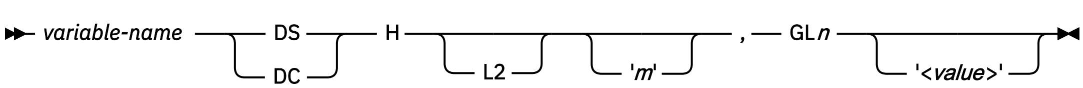
- Fixed-length strings
- Varying-length strings
- DBCLOBs

The following diagrams show the syntax for forms other than DBCLOBs. In the syntax diagrams, *value* denotes one or more DBCS characters, and the symbols < and > represent the shift-out and shift-in characters.

The following diagram shows the syntax for declaring fixed-length graphic strings.



The following diagram shows the syntax for declaring varying-length graphic strings.



Binary host variables

The following diagram shows the syntax for declaring binary host variables.

```
►►─ variable-name ── DS ── X ── Ln ─(1)─►◄
```

Notes:

[1] $1 \le n \le 255$

Varbinary host variables

The following diagram shows the syntax for declaring varbinary host variables.

```
►►─ variable-name ── DS ── H ── L2 ── , ── X ── Ln ─(1)─►◄
```

Notes:

[1] $1 \le n \le 32704$

Result set locators

The following diagram shows the syntax for declaring result set locators.

```
►►─ variable-name ──── SQL TYPE IS RESULT_SET_LOCATOR VARYING ─(1)─►◄
```

Notes:

[1] To be compatible with previous releases, result set locator host variables may be declared as fullword integers (FL4), but the method shown is the preferred syntax.

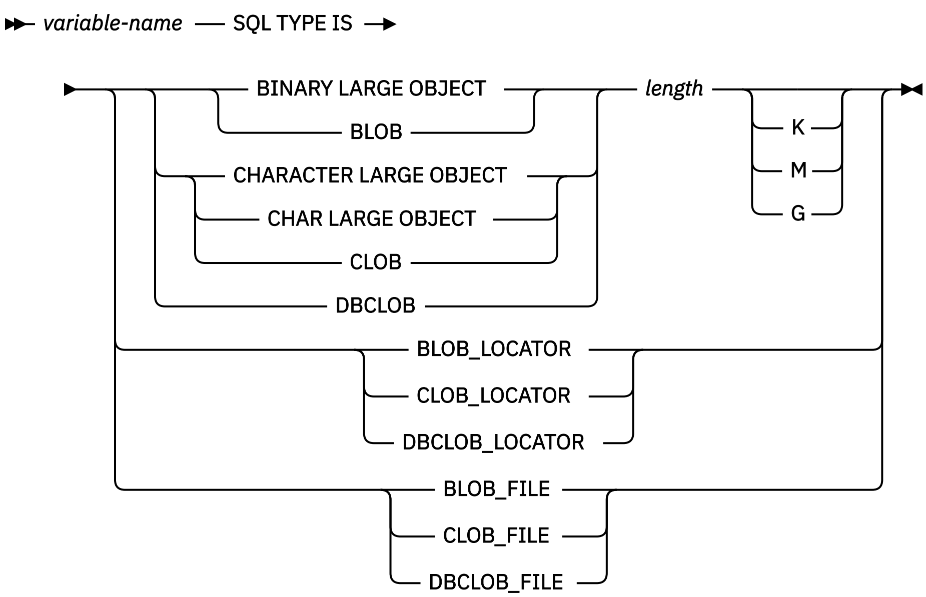
Table Locators

The following diagram shows the syntax for declaring of table locators.

```
►►─ variable-name ── SQL TYPE IS ── TABLE LIKE ── table-name ── AS LOCATOR ─►◄
```

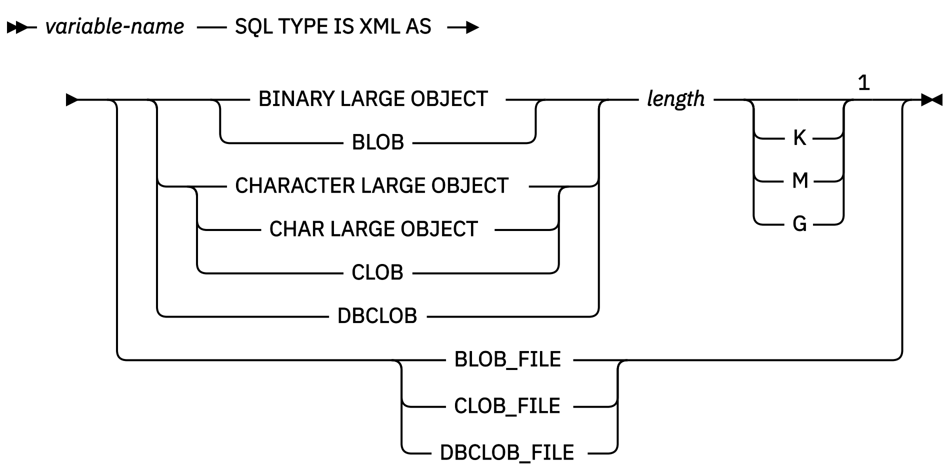
LOB variables, locators, and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables, locators, and file reference variables.



XML data host and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables and file reference variables for XML data types.

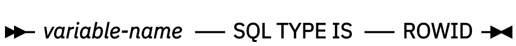


Notes:

[1] If you specify the length of the LOB in terms of KB, MB, or GB, do not leave spaces between the length and K, M, or G.

ROWIDs

The following diagram shows the syntax for declaring ROWID host variables.



Related concepts

Host variables

Use host variables to pass a single data item between Db2 and your application.

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Related tasks

Determining whether a retrieved value in a host variable is null or truncated
Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Inserting a single row by using a host variable
Use host variables in your INSERT statement when you don't know at least some of the values to insert until the program runs.

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Retrieving a single row of data into host variables
If you know that your query returns only one row, you can specify one or more host variables to contain the column values of the retrieved row.

Updating data by using host variables
When you want to update a value in a Db2 table, but you do not know the exact value until the program runs, use host variables. Db2 can change a table value to match the current value of the host variable.

Related reference

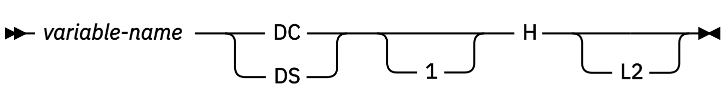
Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

High Level Assembler (HLASM) and Toolkit Feature Library

Indicator variables in assembler

An indicator variable is a 2-byte integer (DS HL2). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

The following diagram shows the syntax for declaring an indicator variable in assembler.



Example

The following example shows a FETCH statement with the declarations of the host variables that are needed for the FETCH statement and their associated indicator variables.

```
    EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,           X
                                   :DAY :DAYIND,      X
                                   :BGN :BGNIND,      X
                                   :END :ENDIND
```

You can declare these variables as follows:

```
CLSCD    DS CL7
DAY      DS HL2
```

```
BGN      DS CL8
END      DS CL8
DAYIND   DS HL2            INDICATOR VARIABLE FOR DAY
BGNIND   DS HL2            INDICATOR VARIABLE FOR BGN
ENDIND   DS HL2            INDICATOR VARIABLE FOR END
```

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related tasks

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Equivalent SQL and assembler data types

When you declare host variables in your assembler programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, ensure that the host variable is of an equivalent data type.

The following table describes the SQL data type and the base SQLTYPE and SQLLEN values that the precompiler uses for host variables in SQL statements.

Table 101. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in assembler programs

Assembler host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
`DS HL2`	500	2	`SMALLINT`
`DS FL4`	496	4	`INTEGER`

`DS P'value'`
`DS PLn'value' or`
`DS PLn`
`1<=n<=16`
	484	*p* in byte 1, *s* in byte 2	`DECIMAL(p,s)`

short decimal FLOAT:
`SDFP DC ED`
`SDFP DC EDL4`
`SDFP DC EDL4'11.11'`
	996	4	`DECFLOAT`

long decimal FLOAT:
`LDFP DC DD`
`LDFP DC DDL8`
`LDFP DC DDL8'22.22'`
	996	8	`DECFLOAT`

extended decimal FLOAT:
`EDFP DC LD`
`EDFP DC LDL16`
`EDFP DC LDL16'33.33'`
	996	16	`DECFLOAT`

`DS EL4`
`DS EHL4`
`DS EBL4`
	480	4	`REAL or FLOAT (n)`
			`1<=n<=21`

Table 101. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in assembler programs (continued)

Assembler host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type

`DS DL8`
`DS DHL8`
`DS DBL8`
	480	8	`DOUBLE PRECISION,`
			`or FLOAT (n)`
			`22<=n<=53`

`DS FDL8` `DS FD`	492	8	`BIGINT`
`SQL TYPE IS BINARY(n)` `1<=n<=255`	912	*n*	`BINARY(n)`

`SQL TYPE IS VARBINARY(n)  or`
`SQL TYPE IS BINARY(n) VARYING`
`1<=n<=32704`
	908	*n*	`VARBINARY(n)`

`DS CLn` `1<=n<=255`	452	*n*	`CHAR(n)`
`DS HL2,CLn` `1<=n<=255`	448	*n*	`VARCHAR(n)`
`DS HL2,CLn` `n>255`	456	*n*	`VARCHAR(n)`

`DS GLm`
`2<=m<=254`
2
	468	*n*	`GRAPHIC(n)`
			3

`DS HL2,GLm`
`2<=m<=254`
2
	464	*n*	`VARGRAPHIC(n)`
			3

`DS HL2,GLm`
`m>254`
2
	472	*n*	`VARGRAPHIC(n)`
			3

`SQL TYPE IS RESULT_SET_LOCATOR`	972	4	`Result set locator`[4,5]

`SQL TYPE IS`
`TABLE LIKE`
`table-name`
`AS LOCATOR`
	976	4	`Table locator`[4]

`SQL TYPE IS` `BLOB_LOCATOR`	960	4	`BLOB locator`[4]
`SQL TYPE IS` `CLOB_LOCATOR`	964	4	`CLOB locator`[4]

Table 101. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in assembler programs (continued)

Assembler host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
`SQL TYPE IS DBCLOB_LOCATOR`	968	4	`DBCLOB locator`[4]
`SQL TYPE IS BLOB(`*n*`) 1≤`*n*`≤2147483647`	404	*n*	`BLOB(`*n*`)`
`SQL TYPE IS CLOB(`*n*`) 1≤`*n*`≤2147483647`	408	*n*	`CLOB(`*n*`)`
`SQL TYPE IS DBCLOB(`*n*`) 1≤`*n*`≤1073741823`	412	*n*	`DBCLOB(`*n*`)` 3
`SQL TYPE IS XML AS BLOB(`*n*`)`	404	0	`XML`
`SQL TYPE IS XML AS CLOB(`*n*`)`	408	0	`XML`
`SQL TYPE IS XML AS DBCLOB(`*n*`)`	412	0	`XML`
`SQL TYPE IS BLOB_FILE`	916/917	267	`BLOB file reference` [4]
`SQL TYPE IS CLOB_FILE`	920/921	267	`CLOB file reference` [4]
`SQL TYPE IS DBCLOB_FILE`	924/925	267	`DBCLOB file reference` [4]
`SQL TYPE IS XML AS BLOB_FILE`	916/917	267	`XML BLOB file reference` [4]
`SQL TYPE IS XML AS CLOB_FILE`	920/921	267	`XML CLOB file reference` [4]
`SQL TYPE IS XML AS DBCLOB_FILE`	924/925	267	`XML DBCLOB file reference` [4]
`SQL TYPE IS ROWID`	904	40	`ROWIDnote 5`

Notes:

1. If a host variable includes an indicator variable, the SQLTYPE value is the base SQLTYPE value plus 1.
2. *m* is the number of bytes.
3. *n* is the number of double-byte characters.
4. This data type cannot be used as a column type.
5. To be compatible with previous releases, result set locator host variables may be declared as fullword integers (FL4), but the method shown is the preferred syntax.

The following table shows equivalent assembler host variables for each SQL data type. Use this table to determine the assembler data type for host variables that you define to receive output from the database. For example, if you retrieve TIMESTAMP data, you can define variable `DS CL`*n*.

This table shows direct conversions between SQL data types and assembler data types. However, a number of SQL data types are compatible. When you do assignments or comparisons of data that have compatible data types, Db2 converts those compatible data types.

Table 102. Assembler host variable equivalents that you can use when retrieving data of a particular SQL data type

SQL data type	**Assembler host variable equivalent**	**Notes**
SMALLINT	DS HL2	
INTEGER	DS F	
BIGINT	DS FD OR DS FDL8	DS FDL8 requires High Level Assembler (HLASM), Release 4 or later.
DECIMAL(*p,s*) or NUMERIC(*p,s*)	DS P'*value*' DS PL*n*'*value*' DS PL*n*	*p* is precision; *s* is scale. 1<=*p*<=31 and 0<=*s*<=*p*. 1<=*n*<=16. *value* is a literal value that includes a decimal point. You must use L*n*, *value*, or both. Using only *value* is recommended. **Precision:** If you use L*n*, it is 2*n*-1; otherwise, it is the number of digits in *value*. *Scale:* If you use *value*, it is the number of digits to the right of the decimal point; otherwise, it is 0. **For efficient use of indexes:** Use *value*. If *p* is *even*, do not use L*n* and be sure the precision of *value* is *p* and the scale of *value* is *s*. If *p* is *odd*, you can use L*n* (although it is not advised), but you must choose *n* so that 2*n*-1=*p*, and *value* so that the scale is *s*. Include a decimal point in *value*, even when the scale of *value* is 0.
REAL or FLOAT(*n*)	DS EL4 DS EHL4 DS EBL4[1]	1<=*n*<=21
DOUBLE PRECISION, DOUBLE, or FLOAT(*n*)	DS DL8 DS DHL8 DS DBL8[1]	22<=*n*<=53
DECFLOAT	DC EDL4 DC DDL8 DC LDL16	
CHAR(*n*)	DS CL*n*	1<=*n*<=255
VARCHAR(*n*)	DS HL2,CL*n*	
GRAPHIC(*n*)	DS GL*m*	*m* is expressed in bytes. *n* is the number of double-byte characters. 1<=*n*<=127
VARGRAPHIC(*n*)	DS HL2,GL*x* DS HL2'*m*',GL*x*'<*value*>'	*x* and *m* are expressed in bytes. *n* is the number of double-byte characters. < and > represent shift-out and shift-in characters.
BINARY(*n*)	**Format 1:** `variable-name--DS--X--Ln` **Format 2:** SQL TYPE IS BINARY(n)	1<=*n*<=255

Table 102. Assembler host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	Assembler host variable equivalent	Notes
VARBINARY(*n*)	**Format 1:** `variable-name--DS--H--L2--,--X--Ln` **Format 2:** SQL TYPE IS VARBINARY(n) or SQL TYPE IS BINARY(n) VARYING	1<=*n*<=32704
DATE	DS CL*n*	If you are using a date exit routine, *n* is determined by that routine; otherwise, *n* must be at least 10.
TIME	DS CL*n*	If you are using a time exit routine, *n* is determined by that routine. Otherwise, *n* must be at least 6; to include seconds, *n* must be at least 8.
TIMESTAMP	DS CL*n*	*n* must be at least 19. To include microseconds, *n* must be 26; if *n* is less than 26, truncation occurs on the microseconds part.
TIMESTAMP(0)	DS CL*n*	*n* must be at least 19.
TIMESTAMP(*p*) *p* > 0	DS CL*n*	*n* must be at least 19. To include fractional seconds, *n* must be 20+*x* where *x* is the number of fractional seconds to include; if *x* is less than *p*, truncation occurs on the fractional seconds part.
TIMESTAMP(0) WITH TIME ZONE	DS HL2,CL*n*	*n* must be at least 25.
TIMESTAMP(*p*) WITH TIME ZONE *p* > 0	DS HL2,CL*n*	*n* must be at least 26+*p*.
Result set locator	DS F	Use this data type only to receive result sets. Do not use this data type as a column type.
Table locator	SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. Do not use this data type as a column type.
BLOB locator	SQL TYPE IS BLOB_LOCATOR	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.
CLOB locator	SQL TYPE IS CLOB_LOCATOR	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB locator	SQL TYPE IS DBCLOB_LOCATOR	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.

Table 102. Assembler host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	**Assembler host variable equivalent**	**Notes**
BLOB(*n*)	SQL TYPE IS BLOB(*n*)	1≤n≤2147483647
CLOB(*n*)	SQL TYPE IS CLOB(*n*)	1≤n≤2147483647
DBCLOB(*n*)	SQL TYPE IS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤n≤1073741823
XML	SQL TYPE IS XML AS BLOB(*n*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS CLOB(*n*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤n≤1073741823
BLOB file reference	SQL TYPE IS BLOB_FILE	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.
CLOB file reference	SQL TYPE IS CLOB_FILE	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB file reference	SQL TYPE IS DBCLOB_FILE	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.
XML BLOB file reference	SQL TYPE IS XML AS BLOB_FILE	Use this data type only to manipulate XML data as BLOB files. Do not use this data type as a column type.
XML CLOB file reference	SQL TYPE IS XML AS CLOB_FILE	Use this data type only to manipulate XML data as CLOB files. Do not use this data type as a column type.
XML DBCLOB file reference	SQL TYPE IS XML AS DBCLOB_FILE	Use this data type only to manipulate XML data as DBCLOB files. Do not use this data type as a column type.
ROWID	SQL TYPE IS ROWID	

Notes:

1. Although stored procedures and user-defined functions can use IEEE floating-point host variables, you cannot declare a user-defined function or stored procedure parameter as IEEE.

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

Host variable data types for XML data in embedded SQL applications (Db2 Programming for XML)

Macros for assembler applications

Data set DSN1110.SDSNMACS contains all Db2 macros that are available for use.

Handling SQL error codes in assembler applications

Assembler applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

To request information about SQL errors in assembler programs, use the following approaches:

- You can use the subroutine DSNTIAR to convert an SQL return code into a text message.

  DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program. For concepts and more information about the behavior of DSNTIAR, see "Displaying SQLCA fields by calling DSNTIAR" on page 529.

  DSNTIAR syntax

  DSNTIAR has the following syntax:

  ```
  CALL DSNTIAR,(sqlca, message, lrecl),MF=(E,PARM)
  ```

  DSNTIAR parameters

  The DSNTIAR parameters have the following meanings:

  sqlca

  An SQL communication area.

  message

  An output area, defined as a varying-length string, in which DSNTIAR places the message text. The first halfword contains the length of the remaining area; its minimum value is 240.

  The output lines of text, each line being the length specified in *lrecl*, are put into this area. For example, you could specify the format of the output area as:

  ```
  LINES    EQU   10
  LRECL    EQU   132
     ⋮
  MSGLRECL DC     AL4(LRECL)
  MESSAGE  DS    H,CL(LINES*LRECL)
           ORG   MESSAGE
  MESSAGEL DC    AL2(LINES*LRECL)
  MESSAGE1 DS    CL(LRECL)      text line 1
  MESSAGE2 DS    CL(LRECL)      text line 2
     ⋮
  MESSAGEn DS    CL(LRECL)      text line n
     ⋮
           CALL DSNTIAR,(SQLCA,MESSAGE,MSGLRECL),MF=(E,PARM)
  ```

  where MESSAGE is the name of the message output area, LINES is the number of lines in the message output area, and LRECL is the length of each line.

  lrecl

  A fullword containing the logical record length of output messages, between 72 and 240.

  The expression MF=(E,PARM) is an z/OS macro parameter that indicates dynamic execution. PARM is the name of a data area that contains a list of pointers to the call parameters of DSNTIAR.

  See "Sample applications supplied with Db2 for z/OS" on page 1010 for instructions on how to access and print the source code for the sample program.

- If your CICS application requires CICS storage handling, you must use the subroutine DSNTIAC instead of DSNTIAR.

  DSNTIAC syntax

  DSNTIAC has the following syntax:

```
CALL DSNTIAC,(eib,commarea,sqlca,msg,lrecl),MF=(E,PARM)
```

DSNTIAC parameters

DSNTIAC has extra parameters, which you must use for calls to routines that use CICS commands.

eib
EXEC interface block

commarea
communication area

For more information on these parameters, see the appropriate application programming guide for CICS. The remaining parameter descriptions are the same as those for DSNTIAR. Both DSNTIAC and DSNTIAR format the SQLCA in the same way.

You must define DSNTIA1 in the CSD. If you load DSNTIAR or DSNTIAC, you must also define them in the CSD. For an example of CSD entry generation statements for use with DSNTIAC, see member DSN8FRDO in the data set *prefix*.SDSNSAMP.

The assembler source code for DSNTIAC and job DSNTEJ5A, which assembles and link-edits DSNTIAC, are also in the data set *prefix*.SDSNSAMP.

- You can also use the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement to convert an SQL return code into a text message.

  Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR. For more information about GET DIAGNOSTICS, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

Related tasks

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference

GET DIAGNOSTICS (Db2 SQL)

C and C++ applications that issue SQL statements

You can code SQL statements in a C or C++ program wherever you can use executable statements.

Each SQL statement in a C or C++ program must begin with EXEC SQL and end with a semicolon (;). The EXEC and SQL keywords must appear on one line, but the remainder of the statement can appear on subsequent lines.

In general, because C is case sensitive, use uppercase letters to enter all SQL keywords. However, if you use the FOLD precompiler suboption, Db2 folds lowercase letters in SBCS SQL ordinary identifiers to uppercase. For information about host language precompiler options, see Table 139 on page 842.

You must keep the case of host variable names consistent throughout the program. For example, if a host variable name is lowercase in its declaration, it must be lowercase in all SQL statements. You might code an UPDATE statement in a C program as follows:

```
EXEC SQL
   UPDATE DSN8B10.DEPT
   SET MGRNO = :mgr_num
   WHERE DEPTNO = :int_dept;
```

Comments

You can include C comments (/* ... */) within SQL statements wherever you can use a blank, except between the keywords EXEC and SQL. You can use single-line comments (starting with //) in C language statements, but not in embedded SQL. You can use SQL comments within embedded SQL statements. For more information, see SQL comments (Db2 SQL).

You can nest comments.

To include EBCDIC DBCS characters in comments, you must delimit the characters by a shift-out and shift-in control character; the first shift-in character in the DBCS string signals the end of the DBCS string.

Continuation for SQL statements
You can use a backslash to continue a character-string constant or delimited identifier on the following line. However, EBCDIC DBCS string constants cannot be continued on a second line.

Delimiters
Delimit an SQL statement in your C program with the beginning keyword EXEC SQL and a Semicolon (;).

Declaring tables and views
Your C program should use the DECLARE TABLE statement to describe each table and view the program accesses. You can use the Db2 declarations generator (DCLGEN) to generate the DECLARE TABLE statements. For more information, see "DCLGEN (declarations generator)" on page 464.

Including SQL statements and variable declarations in source code that is to be processed by the Db2 precompiler
To include SQL statements or C host variable declarations from a member of a partitioned data set, add the following SQL statement to the source code where you want to include the statements:

```
EXEC SQL INCLUDE member-name;
```

You cannot nest SQL INCLUDE statements. Do not use C `#include` statements to include SQL statements or C host variable declarations.

Margins
Code SQL statements in columns 1 through 72, unless you specify other margins to the Db2 precompiler. If EXEC SQL is not within the specified margins, the Db2 precompiler does not recognize the SQL statement. The margin rules do not apply to the Db2 coprocessor. The Db2 coprocessor allows variable length source input.

Names
You can use any valid C name for a host variable, subject to the following restrictions:

- Do not use DBCS characters.
- Do not use external entry names or access plan names that begin with 'DSN', and do not use host variable names or macro names that begin with 'SQL' (in any combination of uppercase or lowercase letters). These names are reserved for Db2.

An SQL identifier that starts with the pound character ('#') can be interpreted as a C macro statement.

Nulls and NULs
C and SQL differ in the way they use the word *null*. The C language has a null character (NUL), a null pointer (NULL), and a null statement (just a semicolon). The C NUL is a single character that compares equal to 0. The C NULL is a special reserved pointer value that does not point to any valid data object. The SQL null value is a special value that is distinct from all non-null values and denotes the absence of a (nonnull) value. NUL (or NUL-terminator) is the null character in C and C++, and NULL is the SQL null value.

Sequence numbers
The Db2 precompiler generates statements without sequence numbers. (The Db2 coprocessor does not perform this action, because the source is read and modified by the compiler.)

Statement labels
You can precede SQL statements with a label.

Trigraph characters
Some characters from the C character set are not available on all keyboards. You can enter these characters into a C source program using a sequence of three characters called a *trigraph*. The trigraph characters that Db2 supports are the same as those that the C compiler supports.

WHENEVER statement
The target for the GOTO clause in an SQL WHENEVER statement must be within the scope of any SQL statements that the statement WHENEVER affects.

Special C/C++ considerations

- Using the C/370 multi-tasking facility, in which multiple tasks execute SQL statements, causes unpredictable results.
- Except for the Db2 coprocessor, you must run the Db2 precompiler before running the C preprocessor.
- Except for the Db2 coprocessor, Db2 precompiler does not support C preprocessor directives.
- If you use conditional compiler directives that contain C code, either place them after the first C token in your application program, or include them in the C program using the `#include` preprocessor directive.

Refer to the appropriate C documentation for more information about C preprocessor directives.

To use the decimal floating-point host data type, you must do the following:

- Use z/OS 1.10 or above (z/OS V1R10 XL C/C++).
- Compile with the C/C++ compiler option, DFP.
- Specify the SQL compiler option to enable the Db2 coprocessor.
- Specify C/C++ compiler option, ARCH(7). It is required by the DFP compiler option if the DFP type is used in the source.
- Specify 'DEFINE(__STDC_WANT_DEC_FP__)' compiler option.

Handling SQL error codes
C and C++ applications can request more information about SQL errors from Db2. For more information, see "Handling SQL error codes in C and C++ applications" on page 617.

Related concepts

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

Related tasks

Overview of programming applications that access Db2 for z/OS data
Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

C and C++ programming examples

You can write Db2 programs in C and C++. These programs can access a local or remote Db2 subsystem and can execute static or dynamic SQL statements. This information contains several such programming examples.

To prepare and run these applications, start with the JCL in member DSNTEJ2D of data set *prefix*.SDSNSAMP as a model for your JCL.

Related concepts

Job DSNTEJ2D (Db2 Installation and Migration)

Related reference

Assembler, C, C++, COBOL, PL/I, and REXX programming examples (Db2 Programming samples)

Sample dynamic and static SQL in a C program

Programs that access Db2 can contain static SQL, dynamic SQL, or both.

This example shows a C program that contains both static and dynamic SQL.

The following figure illustrates dynamic SQL and static SQL embedded in a C program. Each section of the program is identified with a comment. Section 1 of the program shows static SQL; sections 2, 3, and 4 show dynamic SQL. The function of each section is explained in detail in the prologue to the program.

```
/**********************************************************************/
/*  Descriptive name = Dynamic SQL sample using C language          */
/*                                                                  */
/*  Function = To show examples of the use of dynamic and static    */
/*             SQL.                                                 */
/*                                                                  */
/*  Notes = This example assumes that the EMP and DEPT tables are   */
/*          defined. They need not be the same as the DB2 Sample    */
/*          tables.                                                 */
/*                                                                  */
/*  Module type    = C program                                      */
/*     Processor   = DB2  precompiler, C compiler                   */
/*     Module size = see link edit                                  */
/*     Attributes  = not reentrant or reusable                      */
/*                                                                  */
/*     Input    =                                                   */
/*                                                                  */
/*                symbolic label/name = DEPT                        */
/*                description = arbitrary table                     */
/*                symbolic label/name = EMP                         */
/*                description = arbitrary table                     */
/*                                                                  */
/*     Output   =                                                   */
/*                                                                  */
/*                symbolic label/name = SYSPRINT                    */
/*                description = print results via printf            */
/*                                                                  */
/*  Exit-normal = return code 0 normal completion                   */
/*                                                                  */
/*  Exit-error =                                                    */
/*                                                                  */
/*     Return code    = SQLCA                                       */
/*                                                                  */
/*     Abend codes    =  none                                       */
/*                                                                  */
/*  External references  = none                                     */
/*                                                                  */
/*     Control-blocks    =                                          */
/*           SQLCA    - sql communication area                      */
/*                                                                  */
/* Logic specification:                                             */
/*                                                                  */
/* There are four SQL sections.                                     */
/*                                                                  */
/* 1) STATIC SQL  1: using static cursor with a SELECT statement.   */
/*    Two output host variables.                                    */
/* 2) Dynamic SQL 2: Fixed-list SELECT, using same SELECT statement */
/*    used in SQL 1 to show the difference. The prepared string     */
/*    :iptstr can be assigned with other dynamic-able SQL statements.*/
/* 3) Dynamic SQL 3: Insert with parameter markers.                 */
/*    Using four parameter markers which represent four input host  */
/*    variables within a host structure.                            */
/* 4) Dynamic SQL 4: EXECUTE IMMEDIATE                              */
/*    A GRANT statement is executed immediately by passing it to DB2 */
/*    via a varying string host variable. The example shows how to  */
/*    set up the host variable before passing it.                   */
/*                                                                  */
/**********************************************************************/

#include "stdio.h"
#include "stdefs.h"
 EXEC SQL INCLUDE SQLCA;
 EXEC SQL INCLUDE SQLDA;
 EXEC SQL BEGIN DECLARE SECTION;
 short edlevel;
 struct { short len;
          char  x1??(56??);
        } stmtbf1, stmtbf2, inpstr;
 struct { short len;
```

```
         char  x1??(15??);
       } lname;
short hv1;
struct { char  deptno??(4??);
         struct { short len;
                  char  x??(36??);
                 } deptname;
         char  mgrno??(7??);
         char  admrdept??(4??);
         char  location??(17??);
       } hv2;
short ind??(4??);
EXEC SQL END   DECLARE SECTION;
EXEC SQL DECLARE EMP TABLE
               (EMPNO          CHAR(6)             ,
                FIRSTNAME      VARCHAR(12)         ,
                MIDINIT        CHAR(1)             ,
                LASTNAME       VARCHAR(15)         ,
                WORKDEPT       CHAR(3)             ,
                PHONENO        CHAR(4)             ,
                HIREDATE       DECIMAL(6)          ,
                JOBCODE        DECIMAL(3)          ,
                EDLEVEL        SMALLINT            ,
                SEX            CHAR(1)             ,
                BIRTHDATE      DECIMAL(6)          ,
                SALARY         DECIMAL(8,2)        ,
                FORFNAME       VARGRAPHIC(12)      ,
                FORMNAME       GRAPHIC(1)          ,
                FORLNAME       VARGRAPHIC(15)      ,
                FORADDR        VARGRAPHIC(256) )   ;
    EXEC SQL DECLARE DEPT TABLE
                (
                DEPTNO         CHAR(3)             ,
                DEPTNAME       VARCHAR(36)         ,
                MGRNO          CHAR(6)             ,
                ADMRDEPT       CHAR(3)             ,
                LOCATION       CHAR(16));
main ()
{
printf("??/n***      begin of program                        ***");
EXEC SQL WHENEVER SQLERROR GO TO HANDLERR;
EXEC SQL WHENEVER SQLWARNING GO TO HANDWARN;
EXEC SQL WHENEVER NOT FOUND GO TO NOTFOUND;
/******************************************************************/
/* Assign values to host variables which will be input to DB2      */
/******************************************************************/
strcpy(hv2.deptno,"M92");
strcpy(hv2.deptname.x,"DDL");
hv2.deptname.len = strlen(hv2.deptname.x);
strcpy(hv2.mgrno,"000010");
strcpy(hv2.admrdept,"A00");
/******************************************************************/
/* Static  SQL 1:  DECLARE CURSOR, OPEN, FETCH, CLOSE             */
/* Select into :edlevel, :lname                                   */
/******************************************************************/
printf("??/n***      begin declare                           ***");
EXEC SQL DECLARE C1 CURSOR FOR SELECT EDLEVEL, LASTNAME FROM EMP
         WHERE EMPNO = '000010';
printf("??/n***      begin open                              ***");
EXEC SQL OPEN C1;

printf("??/n***      begin fetch                             ***");
EXEC SQL FETCH C1 INTO :edlevel, :lname;
printf("??/n*** returned values                              ***");
printf("??/n??/nedlevel = %d",edlevel);
printf("??/nlname = %s\n",lname.x1);

printf("??/n***      begin close                             ***");
EXEC SQL CLOSE C1;
/******************************************************************/
/* Dynamic SQL 2:  PREPARE, DECLARE CURSOR, OPEN, FETCH, CLOSE    */
/* Select into :edlevel, :lname                                   */
/******************************************************************/
sprintf (inpstr.x1,
         "SELECT EDLEVEL, LASTNAME FROM EMP WHERE EMPNO = '000010'");
inpstr.len = strlen(inpstr.x1);
printf("??/n***      begin prepare                           ***");
EXEC SQL PREPARE STAT1 FROM :inpstr;
printf("??/n***      begin declare                           ***");
EXEC SQL DECLARE C2 CURSOR FOR STAT1;
printf("??/n***      begin open                              ***");
EXEC SQL OPEN C2;
```

```
  printf("??/n***      begin fetch                              ***");
  EXEC SQL FETCH C2 INTO :edlevel, :lname;
  printf("??/n*** returned values                              ***");
  printf("??/n??/nedlevel = %d",edlevel);
  printf("??/nlname = %s??/n",lname.x1);

  printf("??/n***      begin close                              ***");
  EXEC SQL CLOSE C2;
  /*****************************************************************/
  /* Dynamic SQL 3:  PREPARE with parameter markers                 */
  /* Insert into with five values.                                 */
  /*****************************************************************/
  sprintf (stmtbf1.x1,
          "INSERT INTO DEPT VALUES (?,?,?,?,?)");
  stmtbf1.len = strlen(stmtbf1.x1);
  printf("??/n***      begin prepare                            ***");
  EXEC SQL PREPARE s1 FROM :stmtbf1;
  printf("??/n***      begin execute                            ***");
  EXEC SQL EXECUTE s1 USING :hv2:ind;
  printf("??/n***   following are expected insert results      ***");
  printf("??/n  hv2.deptno = %s",hv2.deptno);
  printf("??/n  hv2.deptname.len = %d",hv2.deptname.len);
  printf("??/n  hv2.deptname.x = %s",hv2.deptname.x);
  printf("??/n  hv2.mgrno = %s",hv2.mgrno);
  printf("??/n  hv2.admrdept = %s",hv2.admrdept);
  printf("??/n  hv2.location = %s",hv2.location);
  EXEC SQL COMMIT;
  /*****************************************************************/
  /* Dynamic SQL 4:  EXECUTE IMMEDIATE                             */
  /* Grant select                                                  */
  /*****************************************************************/
  sprintf (stmtbf2.x1,
          "GRANT SELECT ON EMP TO USERX");
  stmtbf2.len = strlen(stmtbf2.x1);
  printf("??/n***      begin execute immediate                  ***");
  EXEC SQL EXECUTE IMMEDIATE :stmtbf2;
  printf("??/n***      end of program                           ***");
  goto progend;
  HANDWARN: HANDLERR: NOTFOUND: ;
  printf("??/n  SQLCODE  = %d",SQLCODE);
  printf("??/n  SQLWARN0 = %c",SQLWARN0);
  printf("??/n  SQLWARN1 = %c",SQLWARN1);
  printf("??/n  SQLWARN2 = %c",SQLWARN2);
  printf("??/n  SQLWARN3 = %c",SQLWARN3);
  printf("??/n  SQLWARN4 = %c",SQLWARN4);
  printf("??/n  SQLWARN5 = %c",SQLWARN5);
  printf("??/n  SQLWARN6 = %c",SQLWARN6);
  printf("??/n  SQLWARN7 = %c",SQLWARN7);
  printf("??/n  SQLERRMC = %s",sqlca.sqlerrmc);
  progend: ;
}
```

Example C program that calls a stored procedure

You can call the C language version of the GETPRML stored procedure that uses the GENERAL WITH NULLS linkage convention.

Because the stored procedure returns result sets, this program checks for result sets and retrieves the contents of the result sets. The following figure contains the example C program that calls the GETPRML stored procedure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
  /************************************************************/
  /* Include the SQLCA and SQLDA                              */
  /************************************************************/
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL INCLUDE SQLDA;
  /************************************************************/
  /* Declare variables that are not SQL-related.              */
  /************************************************************/
  short int i;                /* Loop counter                 */
  /************************************************************/
  /* Declare the following:                                   */
```

```
  /* - Parameters used to call stored procedure GETPRML       */
  /* - An SQLDA for DESCRIBE PROCEDURE                        */
  /* - An SQLDA for DESCRIBE CURSOR                           */
  /* - Result set variable locators for up to three result    */
  /*   sets                                                   */
  /**********************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
    char procnm[19];          /* INPUT parm -- PROCEDURE name */
    char schema[9];           /* INPUT parm -- User's schema  */
    long int out_code;        /* OUTPUT -- SQLCODE from the   */
                              /*   SELECT operation.          */
    char parmlst[255];        /* OUTPUT -- RUNOPTS values     */
                              /*   for the matching row in    */
                              /*   catalog table SYSROUTINES  */
    struct indicators {
      short int procnm_ind;
      short int schema_ind;
      short int out_code_ind;
      short int parmlst_ind;
      } parmind;
                              /* Indicator variable structure */

    struct sqlda *proc_da;
                              /* SQLDA for DESCRIBE PROCEDURE */
    struct sqlda *res_da;
                              /* SQLDA for DESCRIBE CURSOR    */
    static volatile
      SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;
                              /* Locator variables            */
  EXEC SQL END DECLARE SECTION;
```

```
 /************************************************************/
 /* Allocate the SQLDAs to be used for DESCRIBE              */
 /* PROCEDURE and DESCRIBE CURSOR.  Assume that at most      */
 /* three cursors are returned and that each result set      */
 /* has no more than five columns.                           */
 /************************************************************/
 proc_da = (struct sqlda *)malloc(SQLDASIZE(3));
 res_da = (struct sqlda *)malloc(SQLDASIZE(5));

  /**********************************************************/
  /* Call the GETPRML stored procedure to retrieve the       */
  /* RUNOPTS values for the stored procedure.   In this      */
  /* example, we request the PARMLIST definition for the     */
  /* stored procedure named DSN8EP2.                         */
  /*                                                         */
  /* The call should complete with SQLCODE +466 because      */
  /* GETPRML returns result sets.                            */
  /**********************************************************/
  strcpy(procnm,"dsn8ep2           ");
                /* Input parameter -- PROCEDURE to be found   */
  strcpy(schema,"        ");
                /* Input parameter -- Schema name for proc    */
  parmind.procnm_ind=0;
  parmind.schema_ind=0;
  parmind.out_code_ind=0;
                /* Indicate that none of the input parameters */
                /* have null values                           */
  parmind.parmlst_ind=-1;
                /* The parmlst parameter is an output parm.   */
                /* Mark PARMLST parameter as null, so the DB2 */
                /* requester does not have to send the entire  */
                /* PARMLST variable to the server.  This      */
                /* helps reduce network I/O time, because     */
                /* PARMLST is fairly large.                   */
  EXEC SQL
    CALL GETPRML(:procnm INDICATOR :parmind.procnm_ind,
                 :schema INDICATOR :parmind.schema_ind,
                 :out_code INDICATOR :parmind.out_code_ind,
                 :parmlst INDICATOR :parmind.parmlst_ind);
  if(SQLCODE!=+466)          /* If SQL CALL failed,           */
  {
                             /*   print the SQLCODE and any   */
                             /*   message tokens              */
      printf("SQL CALL failed due to SQLCODE = %d\n",
               sqlca.sqlcode);
      printf("sqlca.sqlerrmc = ");
      for(i=0;i<sqlca.sqlerrml;i++)
        printf("%c",sqlca.sqlerrmc[i]);
```

```
      printf("\n");
    }
```

```
    else                        /* If the CALL worked,           */
      if(out_code!=0)          /* Did GETPRML hit an error?     */
        printf("GETPRML failed due to RC = %d\n", out_code);
      /**********************************************************/
      /* If everything worked, do the following:                */
      /* - Print out the parameters returned.                   */
      /* - Retrieve the result sets returned.                   */
      /**********************************************************/
      else
      {
        printf("RUNOPTS = %s\n", parmlst);
                                /* Print out the runopts list    */

        /********************************************************/
        /* Use the statement DESCRIBE PROCEDURE to              */
        /* return information about the result sets in the      */
        /* SQLDA pointed to by proc_da:                         */
        /* - SQLD contains the number of result sets that were  */
        /*   returned by the stored procedure.                  */
        /* - Each SQLVAR entry has the following information    */
        /*   about a result set:                                */
        /*   - SQLNAME contains the name of the cursor that     */
        /*     the stored procedure uses to return the result   */
        /*     set.                                             */
        /*   - SQLIND contains an estimate of the number of     */
        /*     rows in the result set.                          */
        /*   - SQLDATA contains the result locator value for    */
        /*     the result set.                                  */
        /********************************************************/
        EXEC SQL DESCRIBE PROCEDURE INTO :*proc_da;
        /********************************************************/
        /* Assume that you have examined SQLD and determined    */
        /* that there is one result set.  Use the statement     */
        /* ASSOCIATE LOCATORS to establish a result set locator */
        /* for the result set.                                  */
        /********************************************************/
        EXEC SQL ASSOCIATE LOCATORS (:loc1) WITH PROCEDURE GETPRML;

        /********************************************************/
        /* Use the statement ALLOCATE CURSOR to associate a     */
        /* cursor for the result set.                           */
        /********************************************************/
        EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
        /********************************************************/
        /* Use the statement DESCRIBE CURSOR to determine the   */
        /* columns in the result set.                           */
        /********************************************************/
        EXEC SQL DESCRIBE CURSOR C1 INTO :*res_da;
```

```
        /********************************************************/
        /* Call a routine (not shown here) to do the following: */
        /* - Allocate a buffer for data and indicator values    */
        /*   fetched from the result table.                     */
        /* - Update the SQLDATA and SQLIND fields in each       */
        /*   SQLVAR of *res_da with the addresses at which to   */
        /*   to put the fetched data and values of indicator    */
        /*   variables.                                         */
        /********************************************************/
        alloc_outbuff(res_da);

        /********************************************************/
        /* Fetch the data from the result table.                */
        /********************************************************/
        while(SQLCODE==0)
          EXEC SQL FETCH C1 USING DESCRIPTOR :*res_da;
      }
    return;
  }
```

Example C stored procedure with a GENERAL linkage convention

You can call a stored procedure that uses the GENERAL linkage convention from a C program.

This example stored procedure does the following:

- Searches the Db2 catalog table SYSROUTINES for a row that matches the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.
- Searches the Db2 catalog table SYSTABLES for all tables in which the value of CREATOR matches the value of input parameter SCHEMA. The stored procedure uses a cursor to return the table names.

The linkage convention used for this stored procedure is GENERAL.

The output parameters from this stored procedure contain the SQLCODE from the SELECT statement and the value of the RUNOPTS column from SYSROUTINES.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL
  STAY RESIDENT NO
  RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 2
  COMMIT ON RETURN NO;
```

The following example is a C stored procedure with linkage convention GENERAL

```
#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/***************************************************************/
/* Declare C variables for SQL operations on the parameters.   */
/* These are local variables to the C program, which you must  */
/* copy to and from the parameter list provided to the stored  */
/* procedure.                                                  */
/***************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
char PROCNM[19];
char SCHEMA[9];
char PARMLST[255];
EXEC SQL END DECLARE SECTION;

/***************************************************************/
/* Declare cursors for returning result sets to the caller.    */
/***************************************************************/
  EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
    SELECT NAME
    FROM SYSIBM.SYSTABLES
    WHERE CREATOR=:SCHEMA;

main(argc,argv)
  int argc;
  char *argv[];
 {
       /********************************************************/
       /* Copy the input parameters into the area reserved in  */
       /* the program for SQL processing.                      */
       /********************************************************/
  strcpy(PROCNM, argv[1]);
  strcpy(SCHEMA, argv[2]);

       /********************************************************/
       /* Issue the SQL SELECT against the SYSROUTINES         */
       /* DB2 catalog table.                                   */
       /********************************************************/
  strcpy(PARMLST, "");              /* Clear PARMLST           */
  EXEC SQL
    SELECT RUNOPTS INTO :PARMLST
        FROM SYSIBM.ROUTINES
```

```
         WHERE NAME=:PROCNM AND
               SCHEMA=:SCHEMA;
```

```
       /********************************************************/
       /* Copy SQLCODE to the output parameter list.           */
       /********************************************************/
  *(int *) argv[3] = SQLCODE;

       /********************************************************/
       /* Copy the PARMLST value returned by the SELECT back to*/
       /* the parameter list provided to this stored procedure.*/
       /********************************************************/
  strcpy(argv[4], PARMLST);

       /********************************************************/
       /* Open cursor C1 to cause DB2 to return a result set   */
       /* to the caller.                                       */
       /********************************************************/
  EXEC SQL OPEN C1;
 }
```

Example C stored procedure with a GENERAL WITH NULLS linkage convention

You can call a stored procedure that uses the GENERAL WITH NULLS linkage convention from a C program.

This example stored procedure does the following:

- Searches the Db2 catalog table SYSROUTINES for a row that matches the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.
- Searches the Db2 catalog table SYSTABLES for all tables in which the value of CREATOR matches the value of input parameter SCHEMA. The stored procedure uses a cursor to return the table names.

The linkage convention for this stored procedure is GENERAL WITH NULLS.

The output parameters from this stored procedure contain the SQLCODE from the SELECT operation, and the value of the RUNOPTS column retrieved from the SYSROUTINES table.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
  RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 2
  COMMIT ON RETURN NO;
```

The following example is a C stored procedure with linkage convention GENERAL WITH NULLS.

```
#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/***************************************************************/
/* Declare C variables used for SQL operations on the          */
/* parameters. These are local variables to the C program,     */
/* which you must copy to and from the parameter list provided */
/* to the stored procedure.                                    */
/***************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
char PROCNM[19];
```

```
char SCHEMA[9];
char PARMLST[255];
struct INDICATORS {
  short int PROCNM_IND;
  short int SCHEMA_IND;
  short int OUT_CODE_IND;
  short int PARMLST_IND;
 } PARM_IND;
EXEC SQL END DECLARE SECTION;

/*****************************************************************/
/* Declare cursors for returning result sets to the caller.    */
/*****************************************************************/
  EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
    SELECT NAME
    FROM SYSIBM.SYSTABLES
    WHERE CREATOR=:SCHEMA;

main(argc,argv)
  int argc;
  char *argv[];
 {

       /********************************************************/
       /* Copy the input parameters into the area reserved in  */
       /* the local program for SQL processing.                */
       /********************************************************/
  strcpy(PROCNM, argv[1]);
  strcpy(SCHEMA, argv[2]);

       /********************************************************/
       /* Copy null indicator values for the parameter list.   */
       /********************************************************/
  memcpy(&PARM_IND,(struct INDICATORS *) argv[5],
    sizeof(PARM_IND));
```

```
       /********************************************************/
       /* If any input parameter is NULL, return an error      */
       /* return code and assign a NULL value to PARMLST.      */
       /********************************************************/
  if (PARM_IND.PROCNM_IND<0 ||
      PARM_IND.SCHEMA_IND<0 || {
    *(int *) argv[3] = 9999;        /* set output return code  */
    PARM_IND.OUT_CODE_IND = 0;      /* value is not NULL       */
    PARM_IND.PARMLST_IND = -1;      /* PARMLST is NULL         */
   }

  else {
       /********************************************************/
       /* If the input parameters are not NULL, issue the SQL  */
       /* SELECT against the SYSIBM.SYSROUTINES catalog        */
       /* table.                                               */
       /********************************************************/
      strcpy(PARMLST, "");          /* Clear PARMLST           */
      EXEC SQL
        SELECT RUNOPTS INTO :PARMLST
            FROM SYSIBM.SYSROUTINES
            WHERE NAME=:PROCNM AND
                  SCHEMA=:SCHEMA;
       /********************************************************/
       /* Copy SQLCODE to the output parameter list.           */
       /********************************************************/
      *(int *) argv[3] = SQLCODE;
      PARM_IND.OUT_CODE_IND = 0;       /* OUT_CODE is not NULL */
    }

       /********************************************************/
       /* Copy the RUNOPTS value back to the output parameter  */
       /* area.                                                */
       /********************************************************/
   strcpy(argv[4], PARMLST);

       /********************************************************/
       /* Copy the null indicators back to the output parameter*/
       /* area.                                                */
       /********************************************************/
   memcpy((struct INDICATORS *) argv[5],&PARM_IND,
     sizeof(PARM_IND));

       /********************************************************/
       /* Open cursor C1 to cause DB2 to return a result set   */
```

```
        /* to the caller.                                     */
        /*****************************************************/
    EXEC SQL OPEN C1;
    }
```

Defining the SQL communications area, SQLSTATE, and SQLCODE in C and C++

C and C++ programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

About this task

If you specify the SQL processing option STDSQL(YES), do not define an SQLCA. If you do, Db2 ignores your SQLCA, and your SQLCA definition causes compile-time errors. If you specify the SQL processing option STDSQL(NO), include an SQLCA explicitly.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Procedure

Choose one of the following actions:

Option	Description
To define the SQL communications area:	a. Code the SQLCA directly in the program or use the following SQL INCLUDE statement to request a standard SQLCA declaration: `EXEC SQL INCLUDE SQLCA` The standard declaration includes both a structure definition and a static data area named 'sqlca'. Db2 sets the SQLCODE and SQLSTATE values in the SQLCA after each SQL statement executes. Your application should check these values to determine whether the last SQL statement was successful.

To declare SQLCODE and SQLSTATE host variables:	a. Declare the SQLCODE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as a long integer:
	`long SQLCODE;`
	b. Declare the SQLSTATE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as a character array of length 6:
	`char SQLSTATE[6];`
	Restriction: Do not declare an SQLSTATE variable as an element of a structure.
	Requirement: After you declare the SQLCODE and SQLSTATE variables, ensure that all SQL statements in the program are within the scope of the declaration of these variables.

Related tasks

Checking the execution of SQL statements

After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas (SQLDA) in C and C++

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Code the SQLDA directly in the program, or use the following SQL INCLUDE statement to request a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA
```

You can place an SQLDA declaration wherever C allows a structure definition. Normal C scoping rules apply. The standard declaration includes only a structure definition with the name sqlda.

Restriction: You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Declaring host variables and indicator variables in C and C++

You can use host variables, host-variable arrays, and host structures in SQL statements in your program to pass data between Db2 and your application.

Procedure

To declare host variables, host-variable arrays, and host structures:

1. Declare the variables according to the following rules and guidelines:
   - You can have more than one host variable declaration section in your program.
   - You can use class members as host variables. Class members that are used as host variables are accessible to any SQL statement within the class. However, you cannot use class objects as host variables.
   - If you specify the ONEPASS SQL processing option, you must explicitly declare each host variable and each host-variable array before using them in an SQL statement. If you specify the TWOPASS

precompiler option, you must declare each host variable before using it in the DECLARE CURSOR statement.

Restriction: The Db2 coprocessor for C/C++ supports only the ONEPASS option.

- If you specify the STDSQL(YES) SQL processing option, you must precede the host language statements that define the host variables and host-variable arrays with the BEGIN DECLARE SECTION statement and follow the host language statements with the END DECLARE SECTION statement. Otherwise, these statements are optional.
- Ensure that any SQL statement that uses a host variable or host-variable array is within the scope of the statement that declares that variable or array.
- If you are using the Db2 precompiler, ensure that the names of host variables and host-variable arrays are unique within the program, even if the variables and variable arrays are in different blocks, classes, procedures, functions, or subroutines. You can qualify the names with a structure name to make them unique.

2. Optional: Define any associated indicator variables, arrays, and structures.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Host variables in C and C++

In C and C++ programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Restrictions:

- Only some of the valid C declarations are valid host variable declarations. If the declaration for a variable is not valid, any SQL statement that references the variable might result in the message UNDECLARED HOST VARIABLE.
- C supports some data types and storage classes with no SQL equivalents, such as register storage class, typedef, and long long.
- The following locator data types are special SQL data types that do not have C equivalents:
  - Result set locator
  - Table locator
  - LOB locators

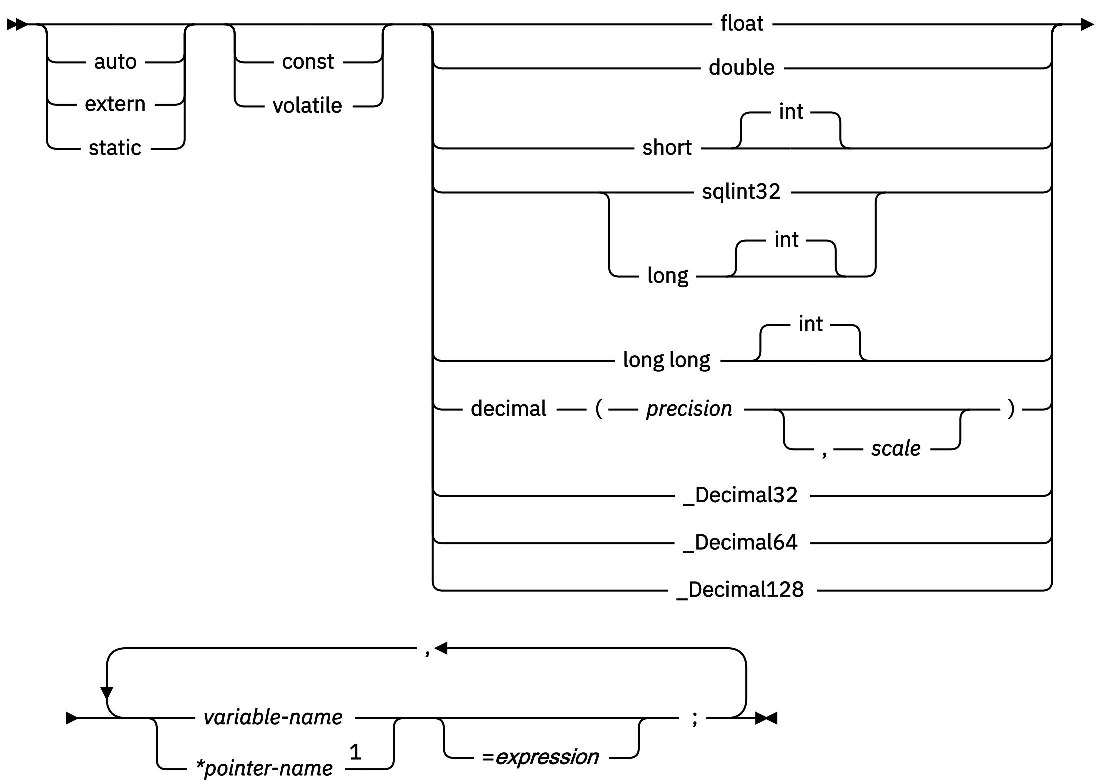
  You cannot use them to define column types.
- Although Db2 allows you to use properly formed L-literals in C application programs, Db2 does not check for all the restrictions that the C compiler imposes on the L-literal. \
- Do not use L-literals in SQL statements. Use Db2 graphic string constants in SQL statements to work with the L-literal.

Recommendations:

- Be careful of overflow. For example, suppose that you retrieve an INTEGER column value into a short integer host variable, and the column value is larger than 32767. You get an overflow warning or an error, depending on whether you provide an indicator variable.
- Be careful of truncation. Ensure that the host variable that you declare can contain the data and a NUL terminator, if needed. Retrieving a floating-point or decimal column value into a long integer host variable removes any fractional part of the value.

Numeric host variables

The following diagram shows the syntax for declaring numeric host variables.



Notes:

[1] If you use the pointer notation of the host variable, you must use the Db2 coprocessor.

Restrictions:

- If your C compiler does not have a decimal data type, no exact equivalent exists for the SQL data type DECIMAL. In this case, you can use one of the following variables or techniques to handle decimal values:
  - An integer or floating-point variable, which converts the value. If you use an integer variable, you lose the fractional part of the number. If the decimal number can exceed the maximum value for an integer or if you want to preserve a fractional value, use floating-point variables. Floating-point numbers are approximations of real numbers. Therefore, when you assign a decimal number to a floating-point variable, the result might be different from the original number.
  - A character-string host variable. Use the CHAR function to get a string representation of a decimal number.
  - The DECIMAL function to explicitly convert a value to a decimal data type, as shown in the following example:

```
long duration=10100;  /* 1 year and 1 month */
char result_dt[11];

EXEC SQL SELECT START_DATE + DECIMAL(:duration,8,0)
         INTO :result_dt FROM TABLE1;
```

- z/OS 1.10 or above (z/OS V1R10 XL C/C++) is required to use the decimal floating-point host data type.
- The special C only 'complex floating-point' host data type is not a supported type for host variable.
- The FLOAT precompiler option does not apply to the decimal floating-point host variable types.
- To use decimal floating-point host variable, you must use the Db2 coprocessor.

For floating-point data types, use the FLOAT SQL processing option to specify whether the host variable is in IEEE binary floating-point or z/Architecture hexadecimal floating-point format. Db2 does not check if the format of the host variable contents match the format that you specified with the FLOAT SQL processing option. Therefore, you need to ensure that your floating-point host variable contents match the format that you specified with the FLOAT SQL processing option. Db2 converts all floating-point input data to z/Architecture hexadecimal floating-point format before storing it.

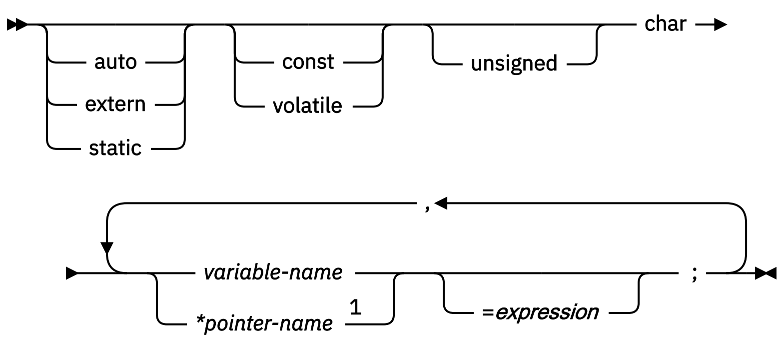
Character host variables

You can specify the following forms of character host variables:

- Single-character form
- NUL-terminated character form
- VARCHAR structured form
- CLOBs

The following diagrams show the syntax for forms other than CLOBs.

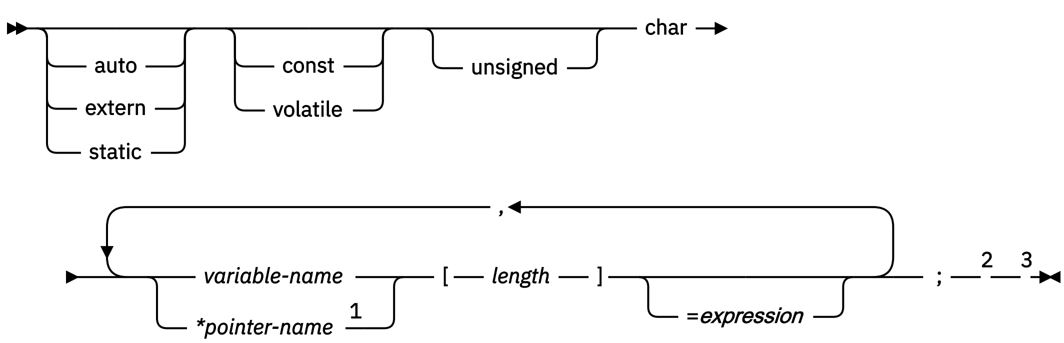
The following diagram shows the syntax for declaring single-character host variables.



Notes:

[1] If you use the pointer notation of the host variable, you must use the Db2 coprocessor.

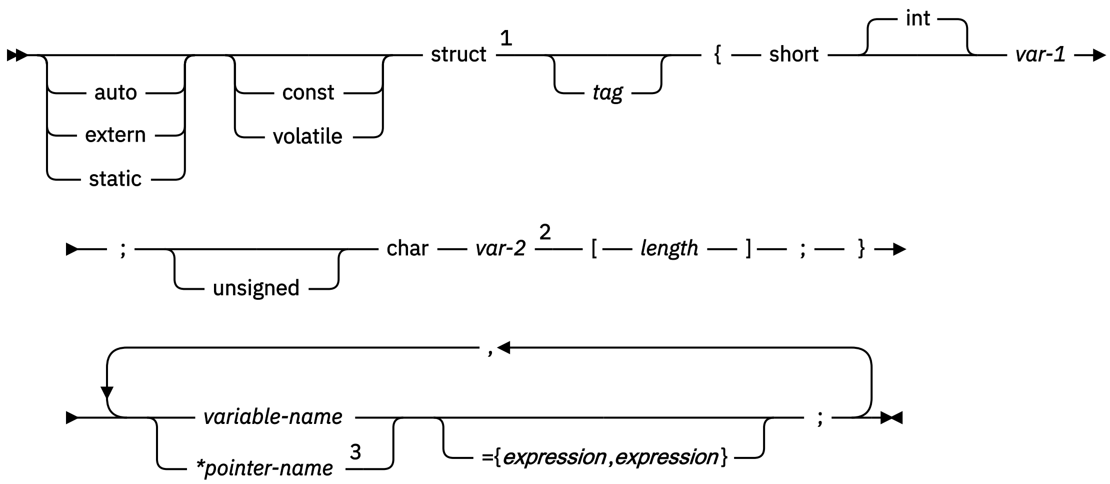
The following diagram shows the syntax for declaring NUL-terminated character host variables.



Notes:

[1] If you use the pointer notation of the host variable, you must use the Db2 coprocessor.
[2] Any string that is assigned to this variable must be NUL-terminated. Any string that is retrieved from this variable is NUL-terminated.
[3] A NUL-terminated character host variable maps to a varying-length character string (except for the NUL).

The following diagram shows the syntax for declaring varying-length character host variables that use the VARCHAR structured form.



Notes:

[1] You can use the struct tag to define other variables, but you cannot use them as host variables in SQL.
[2] You cannot use *var-1* and *var-2* as host variables in an SQL statement.
[3] If you use the pointer notation of the host variable, you must use the Db2 coprocessor.

Example

The following example code shows valid and invalid declarations of the VARCHAR structured form:

```
EXEC SQL BEGIN DECLARE SECTION;

    /* valid declaration of host variable VARCHAR vstring */
    struct VARCHAR {
      short len;
      char s[10];
      } vstring;

    /* invalid declaration of host variable VARCHAR wstring */
    struct VARCHAR wstring;
```

For NUL-terminated string host variables, use the SQL processing options PADNTSTR and NOPADNTSTR to specify whether the variable should be padded with blanks. The option that you specify determines where the NUL-terminator is placed.

If you assign a string of length *n* to a NUL-terminated string host variable, the variable has one of the values that is shown in the following table.

Table 103. Value of a NUL-terminated string host variable that is assigned a string of length n

Length of the NUL-terminated string host variable	Value of the variable
Less than or equal to *n*	The source string up to a length of *n*-1 and a NUL at the end of the string. [1] Db2 sets SQLWARN[1] to W and any indicator variable that you provide to the original length of the source string.
Equal to *n*+1	The source string and a NUL at the end of the string. [1]

Table 103. Value of a NUL-terminated string host variable that is assigned a string of length n (continued)

Length of the NUL-terminated string host variable	Value of the variable
Greater than *n*+1 and the source is a fixed-length string	**If PADNTSTR is in effect** The source string, blanks to pad the value, and a NUL at the end of the string. **If NOPADNTSTR is in effect** The source string and a NUL at the end of the string.
Greater than *n*+1 and the source is a varying-length string	The source string and a NUL at the end of the string. [1]

Note:

1. In these cases, whether NOPADNTSTR or PADNTSTR is in effect is irrelevant.

Restriction: If you use the Db2 precompiler, you cannot use a host variable that is of the NUL-terminated form in either a PREPARE or DESCRIBE statement. However, if you use the Db2 coprocessor, you can use host variables of the NUL-terminated form in PREPARE, DESCRIBE, and EXECUTE IMMEDIATE statements.

Graphic host variables

You can specify the following forms of graphic host variables:

- Single-graphic form
- NUL-terminated graphic form
- VARGRAPHIC structured form.
- DBCLOBs

Recommendation: Instead of using the C data type wchar_t to define graphic and vargraphic host variables, use one of the following techniques:

- Define the sqldbchar data type by using the following typedef statement:

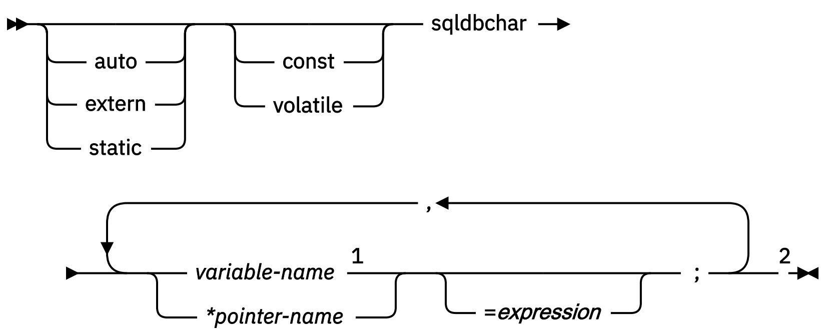
```
typedef unsigned short sqldbchar;
```

- Use the sqldbchar data type that is defined in the typedef statement in one of the following files or libraries:
  - SQL library, sql.h
  - Db2 CLI library, sqlcli.h
  - SQLUDF file in data set DSN1110.SDSNC.H
- Use the C data type unsigned short.

Using sqldbchar or unsigned short enables you to manipulate DBCS and Unicode UTF-16 data in the same format in which it is stored in Db2. Using sqldbchar also makes applications easier to port to other platforms.

The following diagrams show the syntax for forms other than DBCLOBs.

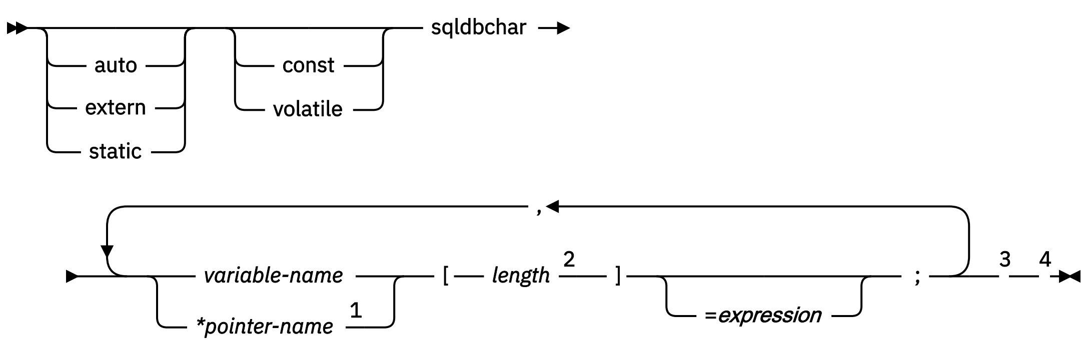
The following diagram shows the syntax for declaring single-graphic host variables.



Notes:

[1] You cannot use array notation in *variable-name*.
[2] The single-graphic form declares a fixed-length graphic string of length 1.

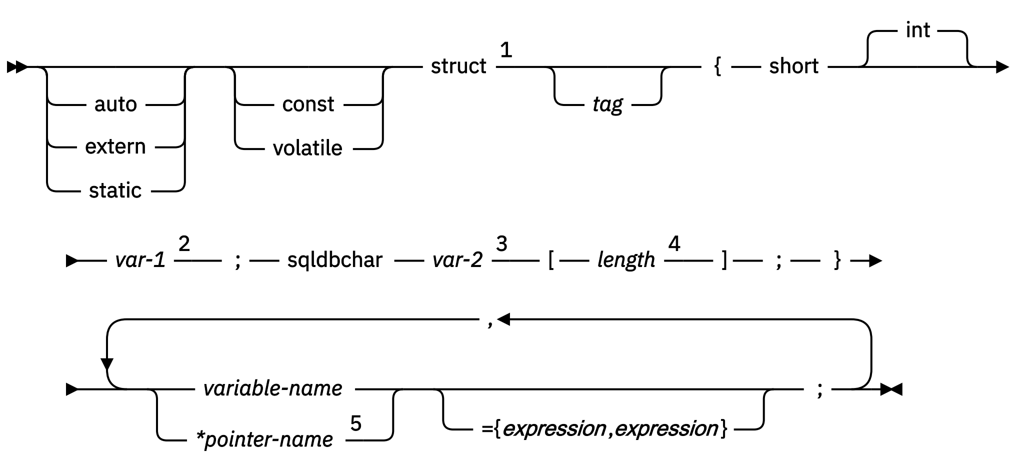
The following diagram shows the syntax for declaring NUL-terminated graphic host variables.



Notes:

[1] If you use the pointer notation of the host variable, you must use the Db2 coprocessor.
[2] *length* must be a decimal integer constant greater than 1 and not greater than 16352.
[3] Any string that is assigned to this variable must be NUL-terminated. Any string that is retrieved from this variable is NUL-terminated.
[4] The NUL-terminated graphic form does not accept single-byte characters for the variable.

The following diagram shows the syntax for declaring graphic host variables that use the VARGRAPHIC structured form.



Notes:

[1] You can use the struct tag to define other variables, but you cannot use them as host variables in SQL.
[2] *var-1* must be less than or equal to *length*.
[3] You cannot use *var-1* or *var-2* as host variables in an SQL statement.
[4] *length* must be a decimal integer constant greater than 1 and not greater than 16352.
[5] f you use the pointer notation of the host variable, you must use the Db2 coprocessor.

Example

The following example shows valid and invalid declarations of graphic host variables that use the VARGRAPHIC structured form:

```
EXEC SQL BEGIN DECLARE SECTION;

   /* valid declaration of host variable structured vgraph */
   struct VARGRAPH {
      short len;
      sqldbchar d[10];
      } vgraph;

   /* invalid declaration of host variable structured wgraph */
   struct VARGRAPH wgraph;
```

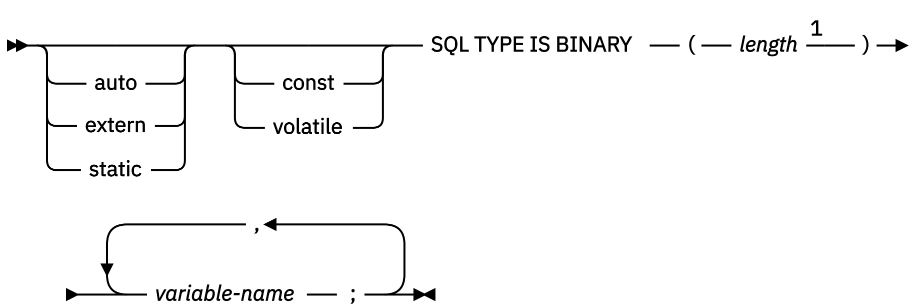
Binary host variables

You can specify the following forms of binary host variables:

- Fixed-length strings
- Varying-length strings
- BLOBs

The following diagrams show the syntax for forms other than BLOBs.

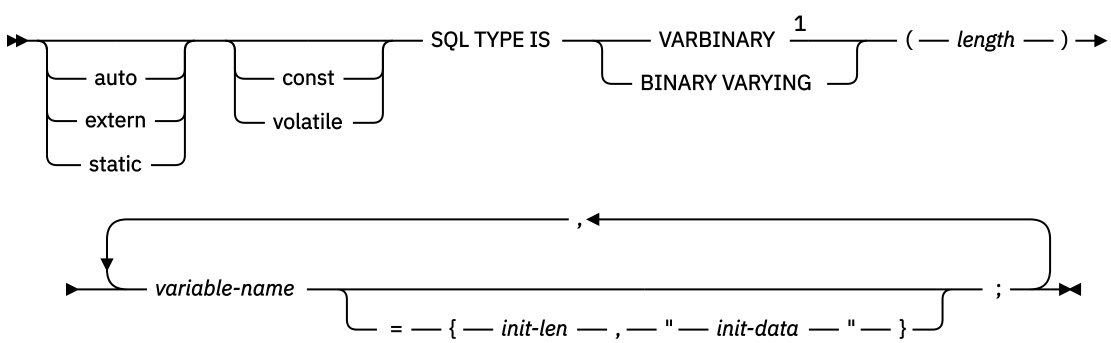
The following diagram shows the syntax for declaring binary host variables.



Notes:

[1] The length must be a value in the range 1 - 255.

The following diagram shows the syntax for declaring VARBINARY host variables.



Notes:

[1] For VARBINARY host variables, the length must be in the range 1 - 32704.

The C language does not have variables that correspond to the SQL binary data types BINARY and VARBINARY. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with the C language structure in the output source member.

When you reference a BINARY or VARBINARY host variable in an SQL statement, you must use the variable that you specify in the SQL TYPE declaration. When you reference the host variable in a host language statement, you must use the variable that Db2 generates.

Examples of binary variable declarations

The following table shows examples of variables that Db2 generates when you declare binary host variables.

Table 104. Examples of BINARY and VARBINARY variable declarations for C

Variable declaration that you include in your C program	Corresponding variable that Db2 generates in the output source member
`SQL TYPE IS BINARY(10) bin_var;`	`char bin_var[10]`
`SQL TYPE IS VARBINARY(10) vbin_var;`	`struct { short length; char data[10]; } vbin_var;`

Recommendation: Be careful when you use binary host variables with C and C++. The SQL TYPE declaration for BINARY and VARBINARY does not account for the NUL-terminator that C expects, because binary strings are not NUL-terminated strings. Also, the binary host variable might contain zeroes at any point in the string.

Result set locators

The following diagram shows the syntax for declaring result set locators.

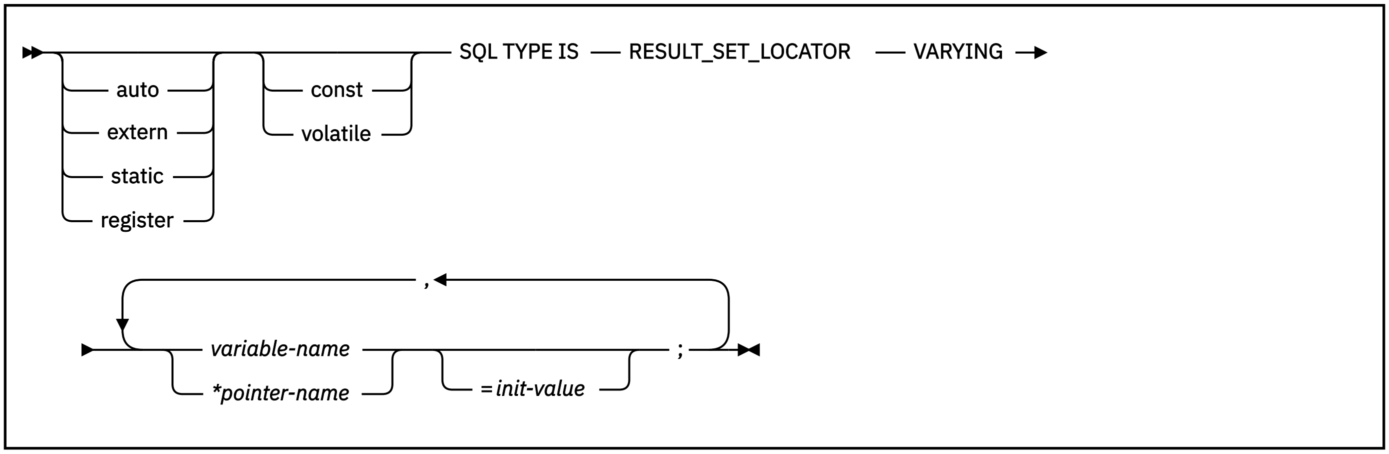
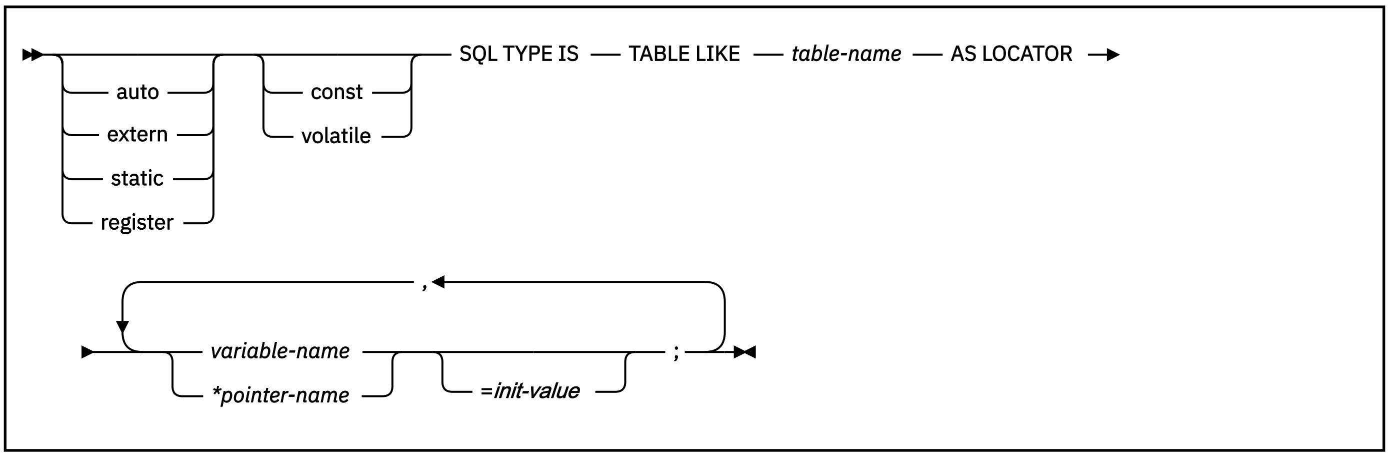


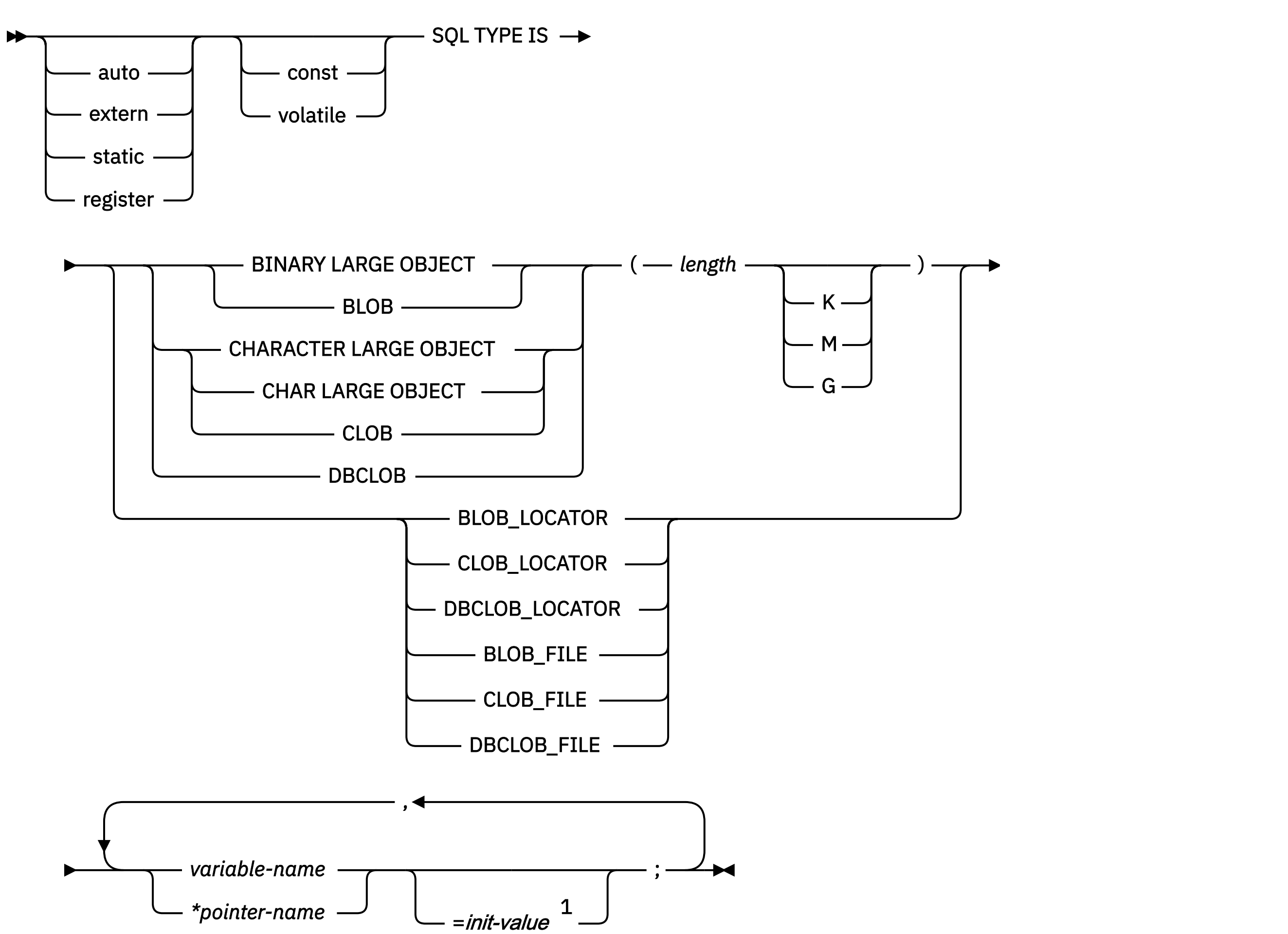
Table locators

The following diagram shows the syntax for declaring table locators.



LOB variables, locators, and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables, locators, and file reference variables.

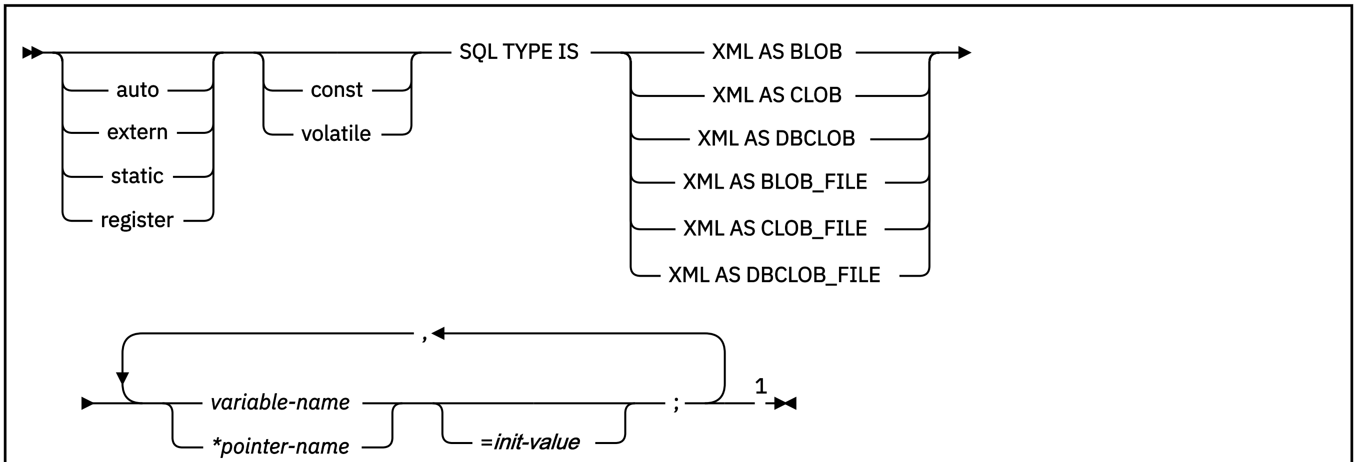


Notes:

[1] Specify the initial value as a series of expressions. For example, specify ={`expression`, `expression`}. For BLOB_FILE, CLOB_FILE, and DBCLOB_FILE, specify ={`name_length`, `data_length`, `file_option_map`, `file_name`}.

XML data host and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables and file reference variables for XML data types.

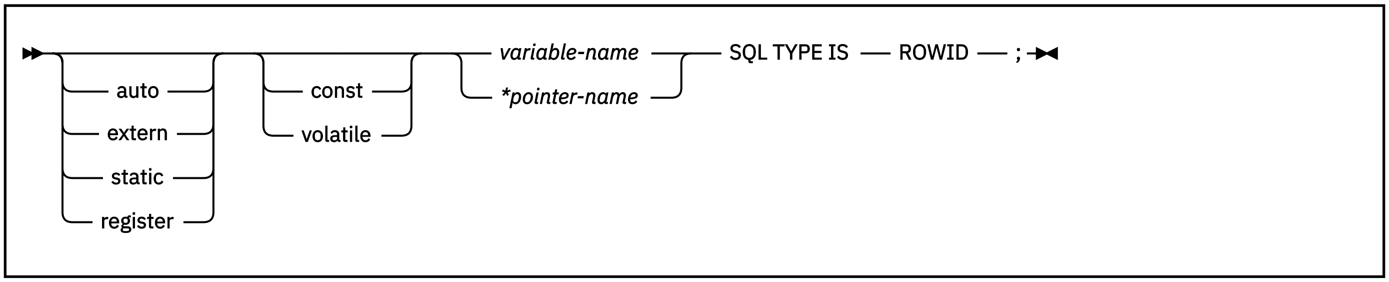


Notes:

[1] Specify the initial value as a series of expressions. For example, specify ={*expression*, *expression*}. For BLOB_FILE, CLOB_FILE, and DBCLOB_FILE, specify ={*name_length*, *data_length*, *file_option_map*, *file_name*}.

ROWID host variables

The following diagram shows the syntax for declaring ROWID host variables.



Constants

The syntax for constants in C and C++ programs differs from the syntax for constants in SQL statements in the following ways:

- C/C++ uses various forms for numeric literals (possible suffixes are: ll, LL, u, U, f,F,l,L,df,DF, dd, DD, dl, DL,d, D). For example, in C/C++:

  4850976 is a decimal literal
  0x4bD is a hexadecimal integer literal
  03245 is an octal integer literal
  3.2E+4 is a double floating-point literal
  3.2E+4f is a float floating-point literal
  3.2E+4l is a long double floating-point literal
  0x4bDP+4 is a double hexadecimal floating-point literal
  22.2df is a _Decimal32 decimal floating-point literal
  0.00D is a fixed-point decimal literal (z/OS only when LANGLVL(EXTENDED) is specified)
- Use C/C++ literal form only outside of SQL statements. Within SQL statements, use numeric constants.
- In C, character constants and string constants can use escape sequences. You cannot use the escape sequences in SQL statements.
- Apostrophes and quotation marks have different meanings in C and SQL. In C, you can use double quotation marks to delimit string constants, and apostrophes to delimit character constants.

  Example: Use of quotation marks in C

  ```
  printf(  "%d lines read. \n", num_lines);
  ```

  Example: Use of apostrophes in C

  ```
  #define NUL '\0'
  ```

  In SQL, you can use double quotation marks to delimit identifiers and apostrophes to delimit string constants.

  Example: quotation marks in SQL

  ```
  SELECT "COL#1" FROM TBL1;
  ```

Example: apostrophes in SQL

```
SELECT COL1 FROM TBL1 WHERE COL2 = 'BELL';
```

- Character data in SQL is distinct from integer data. Character data in C is a subtype of integer data.

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Related tasks

Determining whether a retrieved value in a host variable is null or truncated
Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Inserting a single row by using a host variable
Use host variables in your INSERT statement when you don't know at least some of the values to insert until the program runs.

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Retrieving a single row of data into host variables
If you know that your query returns only one row, you can specify one or more host variables to contain the column values of the retrieved row.

Retrieving a single row of data into a host structure
If you know that your query returns multiple column values for only one row, you can specify a host structure to contain the column values.

Updating data by using host variables
When you want to update a value in a Db2 table, but you do not know the exact value until the program runs, use host variables. Db2 can change a table value to match the current value of the host variable.

Related reference

Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Host-variable arrays in C and C++

In C and C++ programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host-variable arrays can be referenced only as a simple reference in the following contexts. In syntax diagrams, *host-variable-array* designates a reference to a host-variable array.

- In a FETCH statement for a multiple-row fetch. See FETCH (Db2 SQL).
- In an INSERT statement with multiple rows of source data. This is also referred to as multiple-row insert. See INSERT (Db2 SQL).
- In a MERGE statement with multiple rows of source data. See MERGE (Db2 SQL).
- In an EXECUTE statement to provide a value for a parameter marker in a dynamic multi-row INSERT or MERGE. See EXECUTE (Db2 SQL).

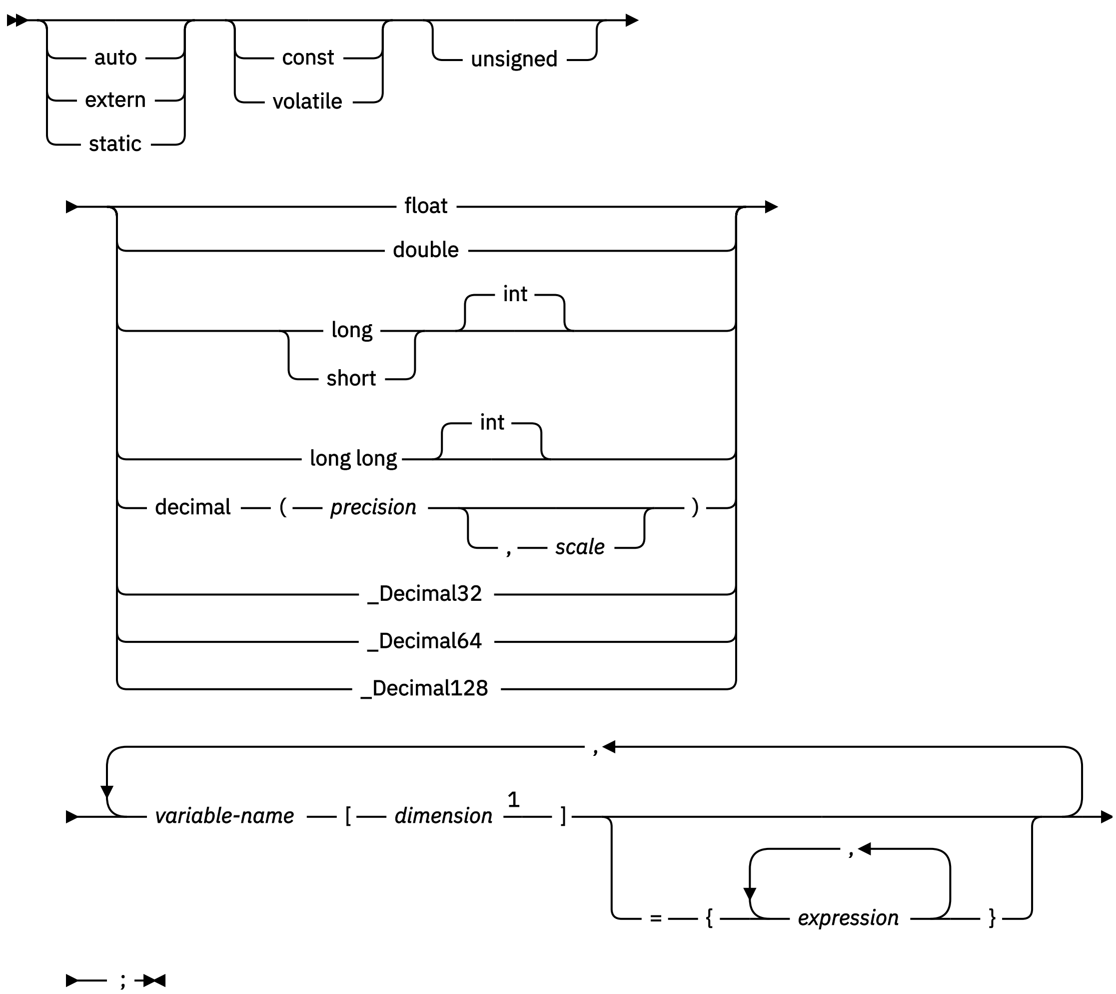
Restrictions:

- Only some of the valid C declarations are valid host-variable array declarations. If the declaration for a variable array is not valid, any SQL statement that references the variable array might result in the message UNDECLARED HOST VARIABLE ARRAY.
- For both C and C++, you cannot specify the _packed attribute on the structure declarations for the following arrays that are used in multiple-row INSERT, FETCH, and MERGE statements:
  - varying-length character arrays
  - varying-length graphic arrays
  - LOB arrays

  In addition, the #pragma pack(1) directive cannot be in effect if you plan to use these arrays in multiple-row statements.

Numeric host-variable arrays

The following diagram shows the syntax for declaring numeric host-variable arrays.



Notes:

1 *dimension* must be an integer constant between 1 and 32767.

Example

The following example shows a declaration of a numeric host-variable array:

```
EXEC SQL BEGIN DECLARE SECTION;
    /* declaration of numeric host-variable array */
    long serial_num[10];
    ...
EXEC SQL END DECLARE SECTION;
```

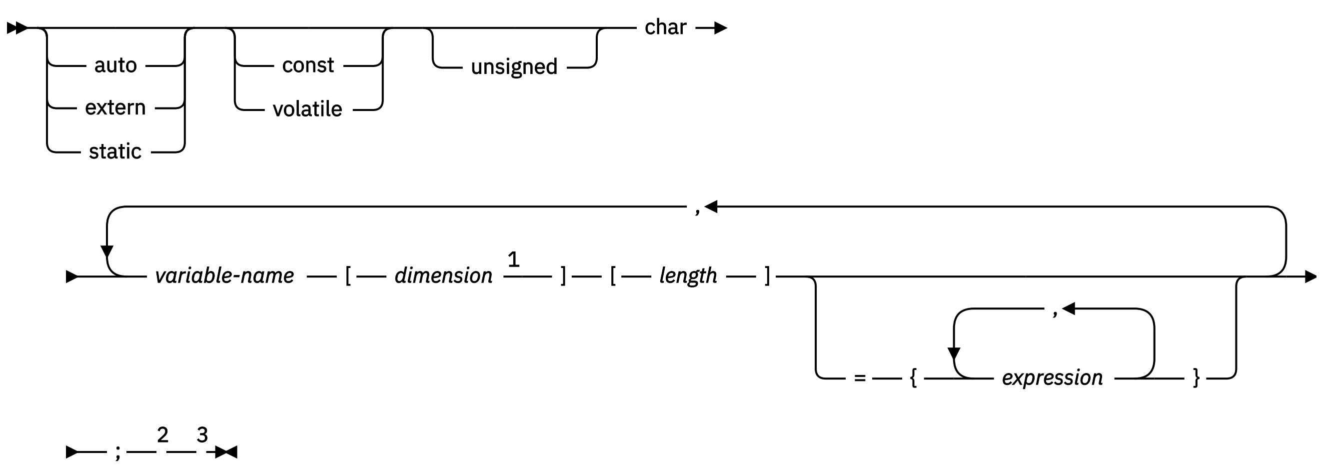
Character host-variable arrays

You can specify the following forms of character host-variable arrays:

- NUL-terminated character form
- VARCHAR structured form
- CLOBs

The following diagrams show the syntax for forms other than CLOBs.

The following diagram shows the syntax for declaring NUL-terminated character host-variable arrays.



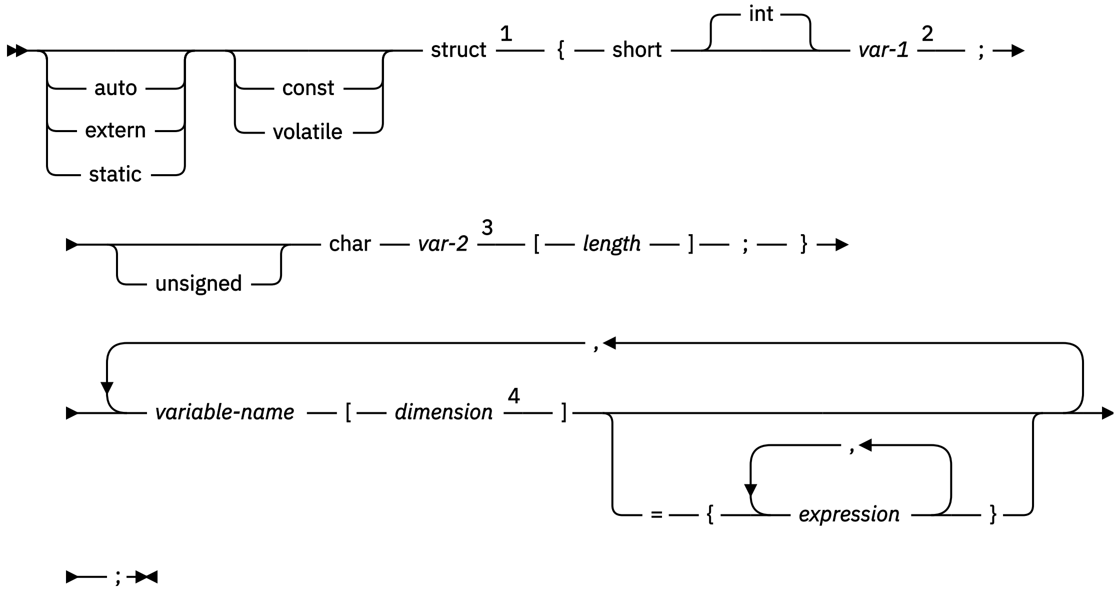
Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

[2] Any string that is assigned to this variable must be NUL-terminated. Any string that is retrieved from this variable is NUL-terminated.

[3] The strings in a NUL-terminated character host-variable array map to varying-length character strings (except for the NUL).

The following diagram shows the syntax for declaring varying-length character host-variable arrays that use the VARCHAR structured form.



Notes:

[1] You can use the struct tag to define other variables, but you cannot use them as host-variable arrays in SQL.
[2] *var-1* must be a scalar numeric variable.
[3] *var-2* must be a scalar CHAR array variable.
[4] *dimension* must be an integer constant between 1 and 32767.

Example

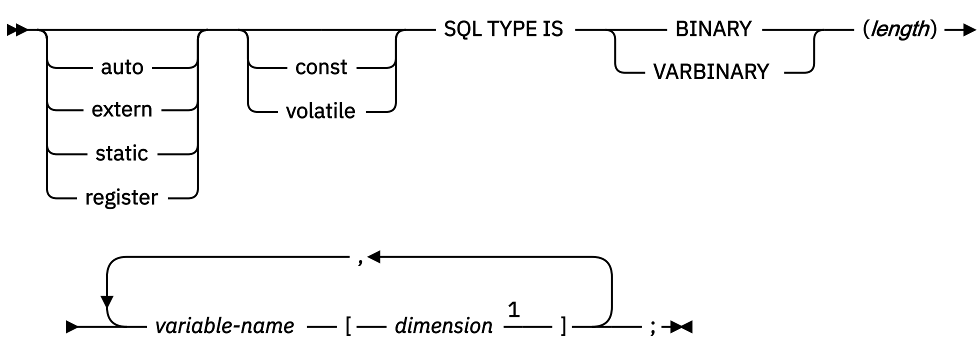
The following example shows valid and invalid declarations of VARCHAR host-variable arrays.

```
EXEC SQL BEGIN DECLARE SECTION;
    /* valid declaration of VARCHAR host-variable array */
    struct VARCHAR {
      short len;
      char s[18];
      } name[10];

/* invalid declaration of VARCHAR host-variable array */
   struct VARCHAR name[10];
```

Binary host-variable arrays

The following diagram shows the syntax for declaring binary host-variable arrays.



Notes:

1 *dimension* must be an integer constant between 1 and 32767.

Graphic host-variable arrays

You can specify the following forms of graphic host-variable arrays:

- NUL-terminated graphic form
- VARGRAPHIC structured form.

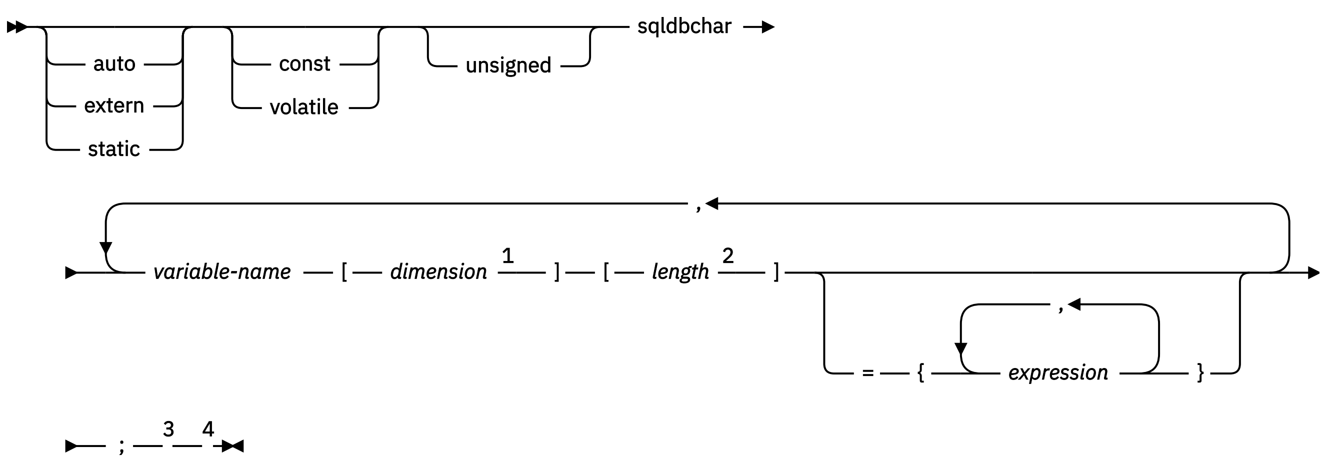
Recommendation: Instead of using the C data type wchar_t to define graphic and vargraphic host-variable arrays, use one of the following techniques:

- Define the sqldbchar data type by using the following typedef statement:

```
typedef unsigned short sqldbchar;
```

- Use the sqldbchar data type that is defined in the typedef statement in the header files that are supplied by Db2.
- Use the C data type unsigned short.

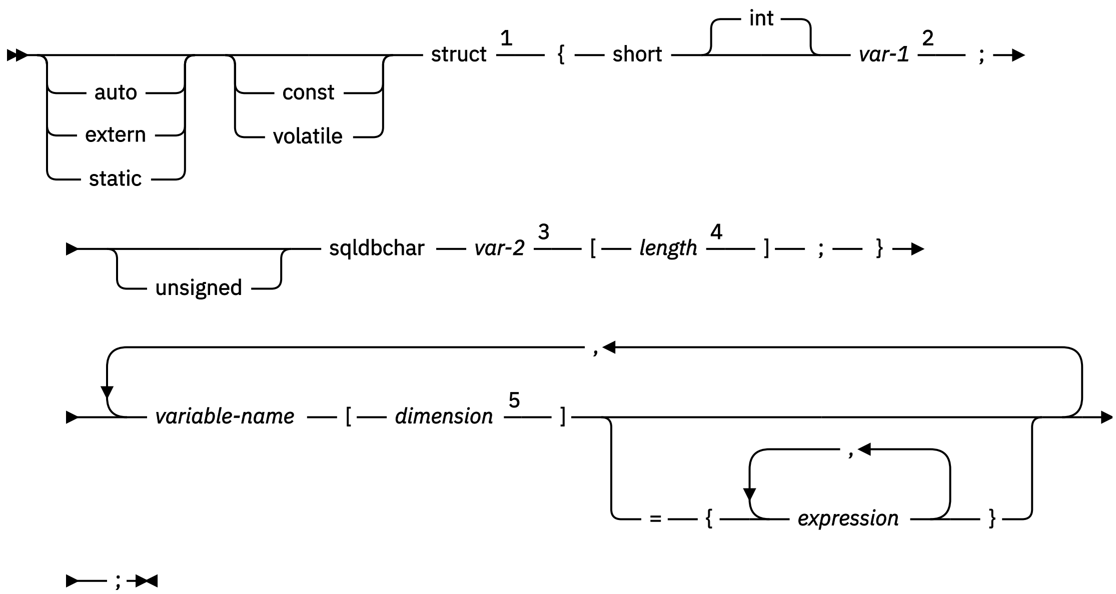
The following diagram shows the syntax for declaring NUL-terminated graphic host-variable arrays.



Notes:

1 *dimension* must be an integer constant between 1 and 32767.
2 *length* must be a decimal integer constant greater than 1 and not greater than 16352.
3 Any string that is assigned to this variable must be NUL-terminated. Any string that is retrieved from this variable is NUL-terminated.
4 Do not assign single-byte characters into a NUL-terminated graphic host-variable array

The following diagram shows the syntax for declaring graphic host-variable arrays that use the VARGRAPHIC structured form.



Notes:

[1] You can use the struct tag to define other variables, but you cannot use them as host-variable arrays in SQL.
[2] *var-1* must be a scalar numeric variable.
[3] *var-2* must be a scalar char array variable.
[4] *length* must be a decimal integer constant greater than 1 and not greater than 16352.
[5] *dimension* must be an integer constant between 1 and 32767.

Example

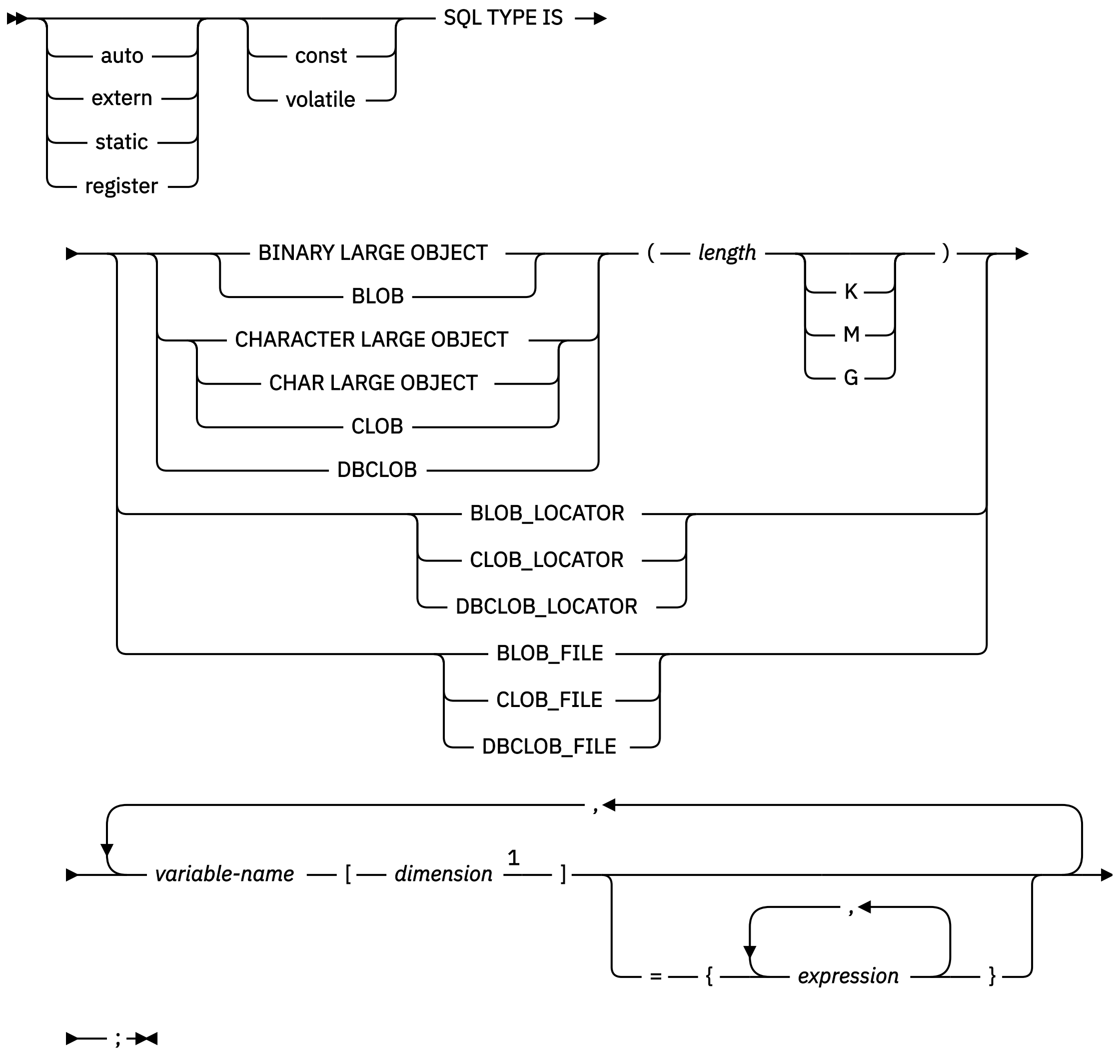
The following example shows valid and invalid declarations of graphic host-variable arrays that use the VARGRAPHIC structured form.

```
EXEC SQL BEGIN DECLARE SECTION;
   /* valid declaration of host-variable array vgraph */
   struct VARGRAPH {
      short len;
      sqldbchar d[10];
      } vgraph[20];

   /* invalid declaration of host-variable array vgraph */
   struct VARGRAPH vgraph[20];
```

LOB, locator, and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host-variable arrays, locators, and file reference variables.

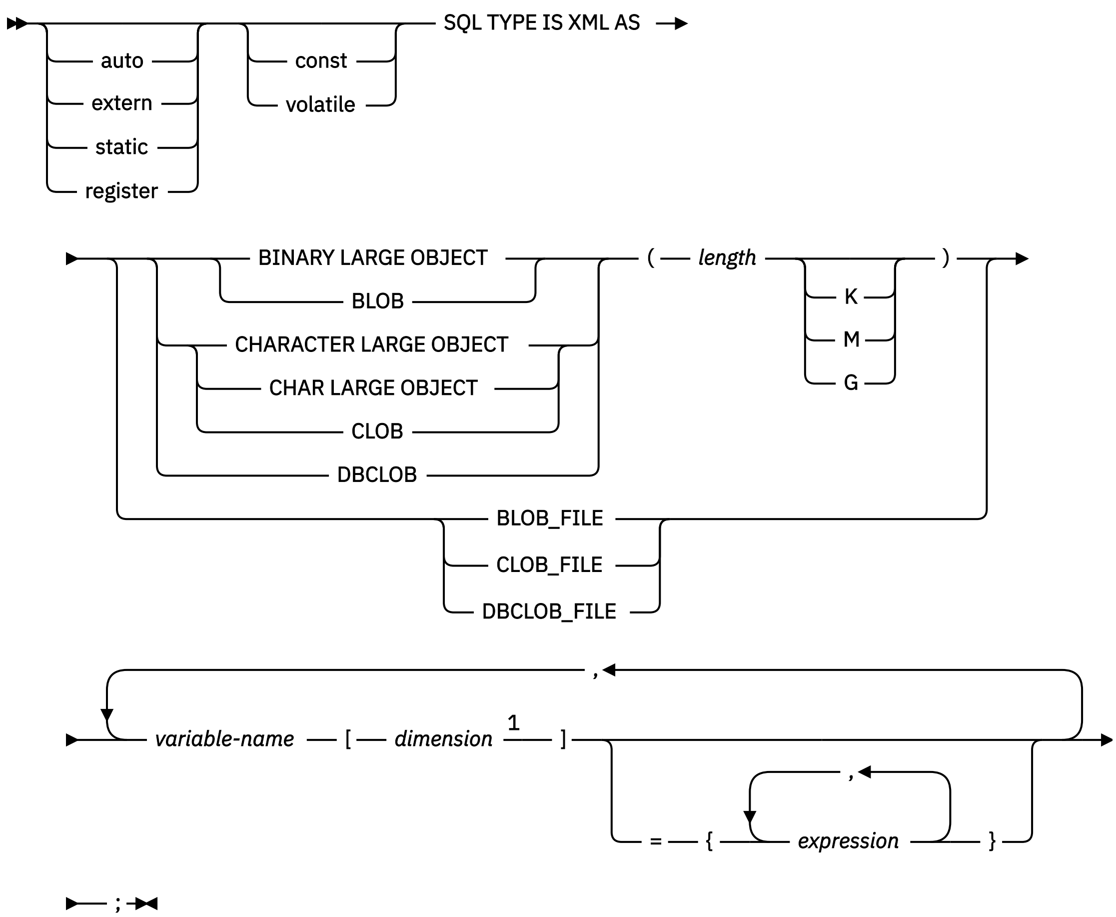


Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

XML host and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host-variable arrays and file reference variable arrays for XML data types.

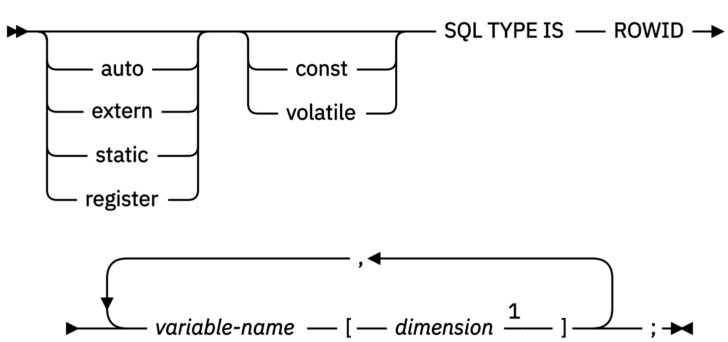


Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

ROWID variable arrays

The following diagram shows the syntax for declaring ROWID variable arrays.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

Related concepts

Using host-variable arrays in SQL statements

Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

Host-variable arrays
You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

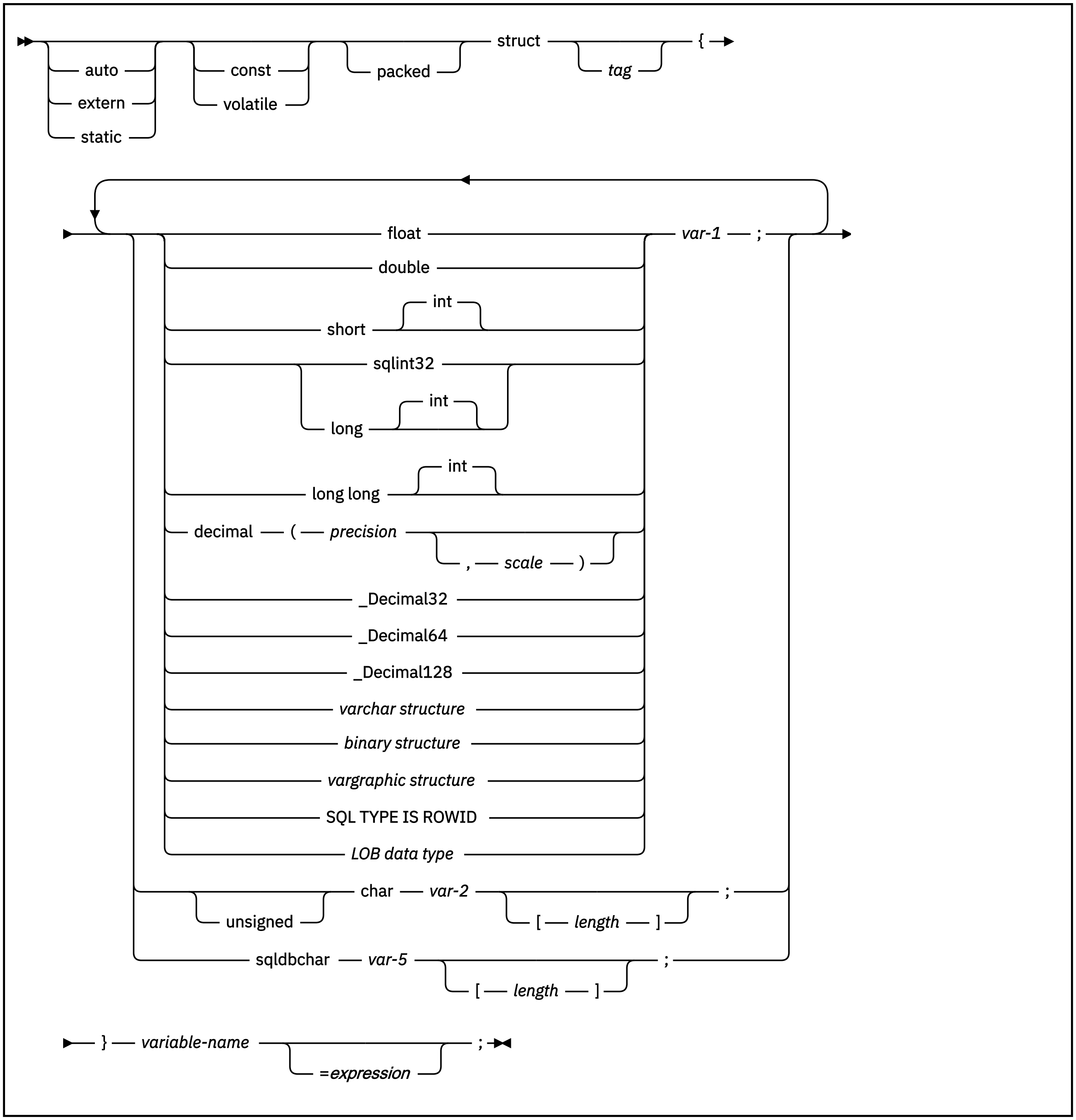
Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Host structures in C and C++

A C host structure contains an ordered group of data fields.

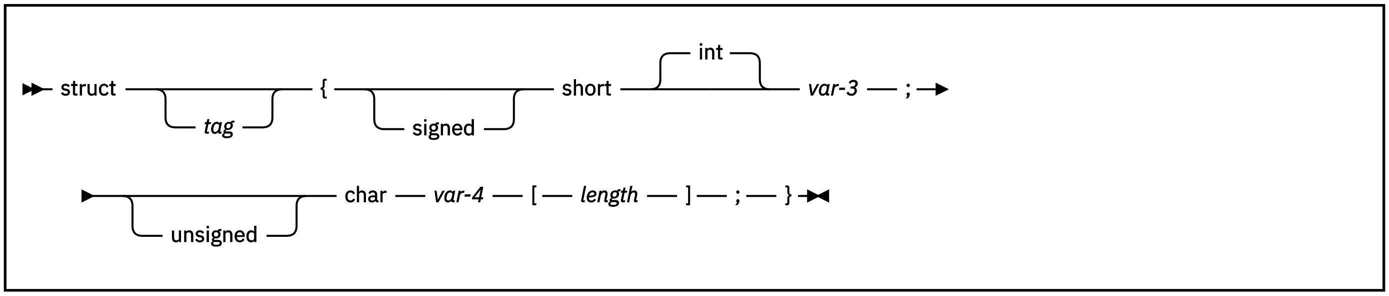
Host structures

The following diagram shows the syntax for declaring host structures.



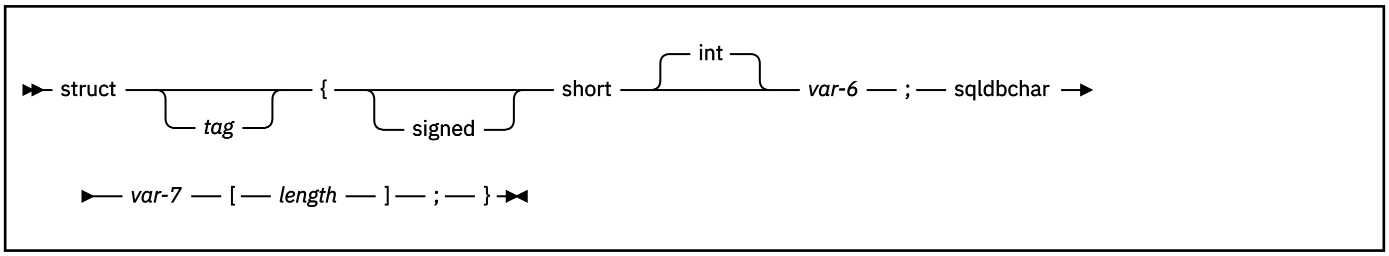
VARCHAR structures

The following diagram shows the syntax for VARCHAR structures that are used within declarations of host structures.



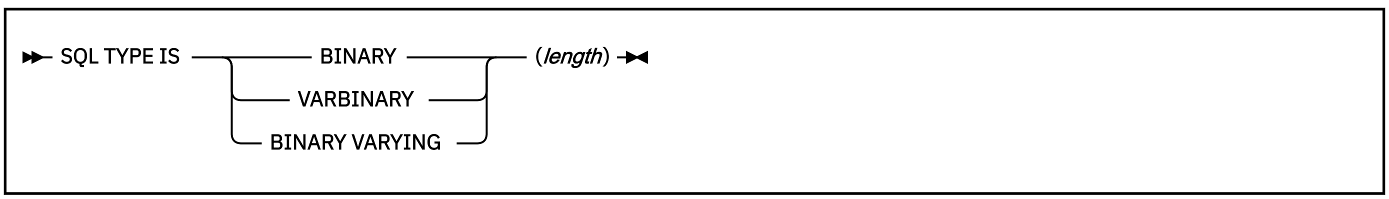
VARGRAPHIC structures

The following diagram shows the syntax for VARGRAPHIC structures that are used within declarations of host structures.



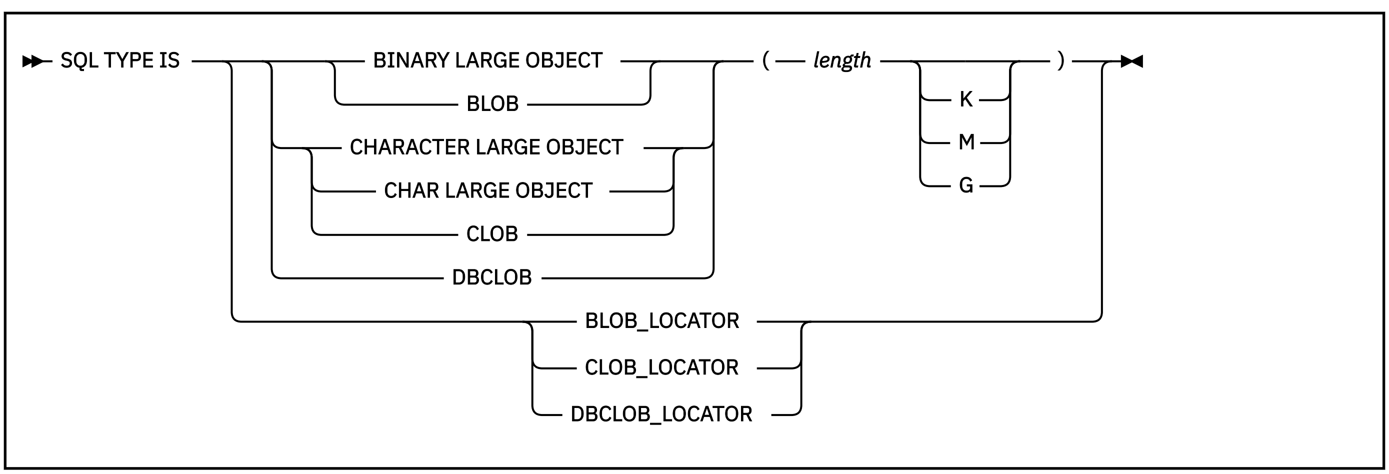
Binary structures

The following diagram shows the syntax for binary structures that are used within declarations of host structures.



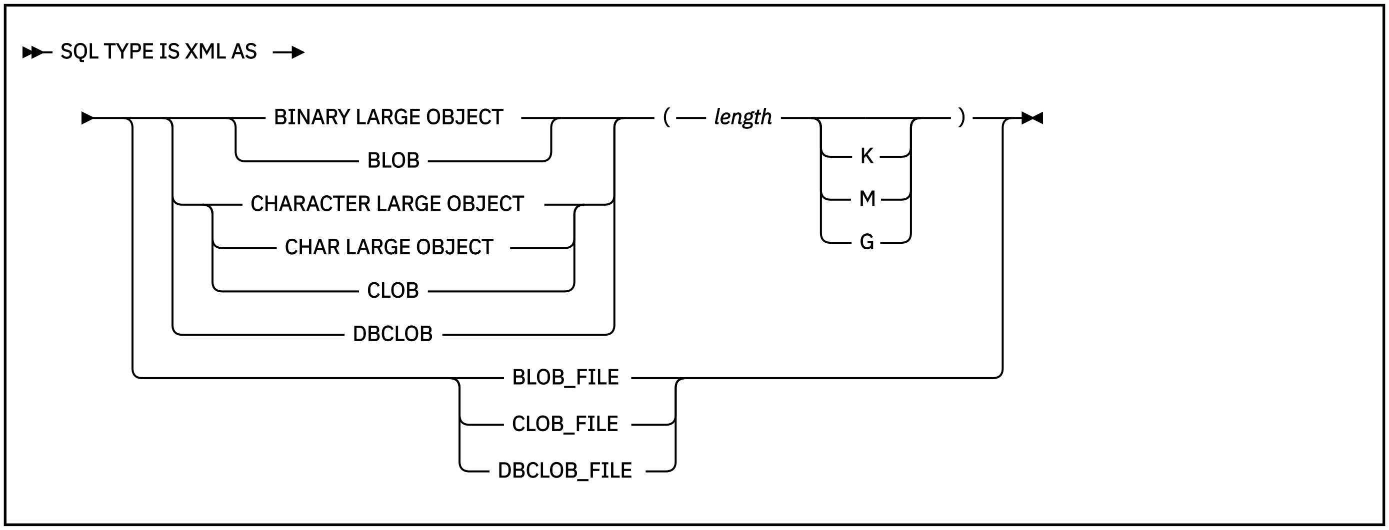
LOB data types

The following diagram shows the syntax for LOB data types that are used within declarations of host structures.



LOB data types for XML data

The following diagram shows the syntax for LOB data types that are used within declarations of host structures for XML data.



Example

In the following example, the host structure is named target, and it contains the fields c1, c2, and c3. c1 and c3 are character arrays, and c2 is a host variable that is equivalent to the SQL VARCHAR data type. The target host structure can be part of another host structure but must be the deepest level of the nested structure.

```
struct {char  c1[3];
        struct {short len;
                char  data[5];
               }c2;
        char  c3[2];
       }target;
```

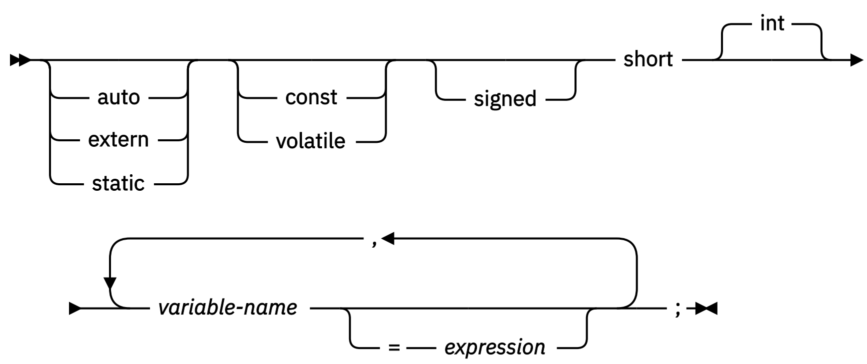
Related concepts

Host structures
Use host structures to pass a group of host variables between Db2 and your application.

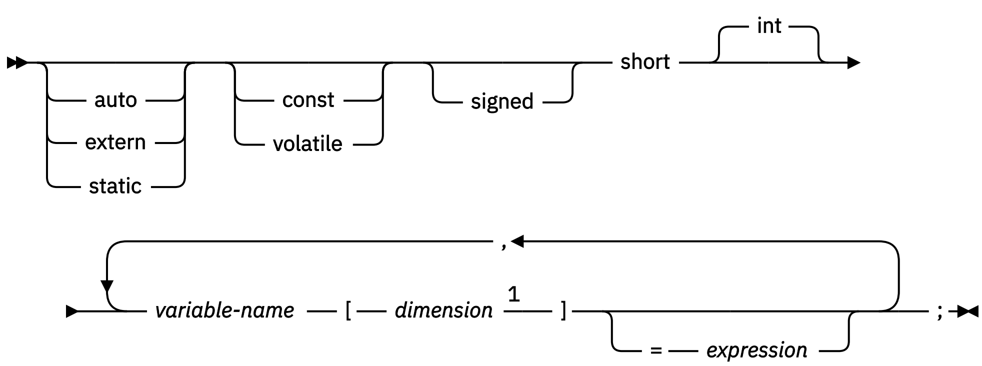
Indicator variables, indicator arrays, and host structure indicator arrays in C and C++

An indicator variable is a 2-byte integer (short int). An indicator variable array is an array of 2-byte integers (short int). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

The following diagram shows the syntax for declaring an indicator variable in C and C++.



The following diagram shows the syntax for declaring an indicator array or a host structure indicator array in C and C++.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

Example

The following example shows a FETCH statement with the declarations of the host variables that are needed for the FETCH statement and their associated indicator variables.

```
EXEC SQL FETCH CLS_CURSOR INTO :ClsCd,
                               :Day :DayInd,
                               :Bgn :BgnInd,
                               :End :EndInd;
```

You can declare these variables as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char  ClsCd[8];
char  Bgn[9];
char  End[9];
short Day, DayInd, BgnInd, EndInd;
EXEC SQL END DECLARE SECTION;
```

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related tasks

Inserting null values into columns by using indicator variables or arrays

If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Referencing pointer host variables in C programs

If you use the Db2 coprocessor, you can reference any declared pointer host variables in your SQL statements.

Procedure

Specify the pointer host variable exactly as it was declared.

The only exception is when you reference pointers to nul-terminated character arrays. In this case, you do not have to include the parentheses that were part of the declaration.

Examples

Examples: Scalar pointer host variable references

Table 105. Example references to scalar pointer host variables

Declaration	Description	Reference
`short *hvshortp;`	hvshortp is a pointer host variable that points to two bytes of storage.	`EXEC SQL set:*hvshortp=123;`
`double *hvdoubp;`	hvdoubp is a pointer host variable that points to eight bytes of storage.	`EXEC SQL set:*hvdoubp=456;`
`char (*hvcharpn) [20];`	hvcharpn is a pointer host variable that points to a nul-terminated character array of up to 20 bytes.	`EXEC SQL set: *hvcharpn='nul_terminated';`

Example:Bounded character pointer host variable reference

Suppose that your program declares the following bounded character pointer host variable:

```
struct {
  unsigned long len;
  char * data;
  } hvbcharp;
```

The following example references this bounded character pointer host variable:

```
hvcharp.len = dynlen; a
hvcharp.data = (char *) malloc (hvcharp.len); b
EXEC SQL set :hvcharp = 'data buffer with length'; c
```

Note:

a
: dynlen can be either a compile time constant or a variable with a value that is assigned at run time.

b
: Storage is dynamically allocated for hvcharp.data.

c
: The SQL statement references the name of the structure, not an element within the structure.

Examples: Array pointer host variable references

Table 106. Example references to array pointer host variables

Declaration	Description	Reference
`short * hvarrpl[6]`	hvarrp1 is an array of 6 pointers that point to two bytes of storage each.	`EXEC SQL set:*hvarrpl[n]=123;`
`double * hvarrp2[3]`	hvarrp2 is an array of 3 pointers that point to 8 bytes of storage each.	`EXEC SQL set:*hvarrp2[n]=456;`
`struct { unsigned long len; char * data; } hvbarrp3[5];`	hvbarrp3 is an array of 5 bounded character pointers.	`EXEC SQL set :hvarrp3[n] = 'data buffer with length'`

Example: Structure array host variable reference

Suppose that your program declares the following pointer to the structure tbl_struct:

```
struct tbl_struct *ptr_tbl_struct =
 (struct tbl_struct *) malloc (sizeof (struct tbl_struct) * n);
```

To reference this data is SQL statements, use the pointer as shown in the following example. Assume that tbl_sel_cur is a declared cursor.

```
for (L_col_cnt = 0; L_col_cnt < n; L_con_cnt++)
{ ...
  EXEC SQL FETCH tbl_sel_cur INTO :ptr_tbl_struct [L_col_cnt]
  ...
}
```

Related tasks

Declaring pointer host variables in C programs
If you use the Db2 coprocessor, you can use pointer host variables with statically or dynamically allocated storage. These pointer host variables can point to numeric data, non-numeric data, or a structure.

Declaring pointer host variables in C programs

If you use the Db2 coprocessor, you can use pointer host variables with statically or dynamically allocated storage. These pointer host variables can point to numeric data, non-numeric data, or a structure.

About this task

You can declare the following types of pointer host variables:

Scalar pointer host variable
A host variable that points to numeric or non-numeric scalar data.

Array pointer host variable
A host variable that is an array of pointers.

Structure array host variable
A host variable that points to a structure.

Procedure

Include an asterisk (*) in each variable declaration to indicate that the variable is a pointer.

Restrictions:

- You cannot use pointer host variables that point to character data of an unknown length. For example, do not specify the following declaration: `char * hvcharpu`. Instead, specify the length of the data by using a bounded character pointer host variable. A *bounded character pointer host variable* is a host variable that is declared as a structure with the following elements:

- A 4-byte field that contains the length of the storage area.
- A pointer to the non-numeric dynamic storage area.

- You cannot use untyped pointers. For example, do not specify the following declaration: `void * untypedprt`.

Examples

Example: Scalar pointer host variable declarations

Table 107. Example declarations of scalar pointer host variables

Declaration	Description
`short *hvshortp;`	hvshortp is a pointer host variable that points to two bytes of storage.
`double *hvdoubp;`	hvdoubp is a pointer host variable that points to eight bytes of storage.
`char (*hvcharpn) [20];`	hvcharpn is a pointer host variable that points to a nul-terminated character array of up to 20 bytes.

Example: Bounded character pointer host variable declaration

The following example code declares a bounded character pointer host variable called hvbcharp with two elements: len and data.

```
struct {
  unsigned long len;
  char * data;
  } hvbcharp;
```

Example: array pointer host variable declarations

Table 108. Example declarations of array pointer host variables

Declaration	Description
`short * hvarrpl[6]`	hvarrp1 is an array of 6 pointers that point to two bytes of storage each.
`double * hvarrp2[3]`	hvarrp2 is an array of 3 pointers that point to 8 bytes of storage each.
`struct { unsigned long len; char * data; } hvbarrp3[5];`	hvbarrp3 is an array of 5 bounded character pointers.

Example: Structure array host variable declaration

The following example code declares a table structure called tbl_struct.

```
struct tbl_struct
{
  char  colname[20];
  small int colno;
  small int coltype;
  small int collen;
};
```

The following example code declares a pointer to the structure tbl_struct. Storage is allocated dynamically for up to n rows.

```
struct tbl_struct *ptr_tbl_struct =
 (struct tbl_struct *) malloc (sizeof (struct tbl_struct) * n);
```

Related tasks

Referencing pointer host variables in C programs
If you use the Db2 coprocessor, you can reference any declared pointer host variables in your SQL statements.

Equivalent SQL and C data types

When you declare host variables in your C programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

The following table describes the SQL data type and the base SQLTYPE and SQLLEN values that the precompiler uses for host variables in SQL statements.

Table 109. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in C programs

C host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
short int	500	2	SMALLINT
long int	496	4	INTEGER
long long long long int sqlint64	492	8	BIGINT[5]
decimal(*p*,*s*)[2]	484	p in byte 1, s in byte 2	DECIMAL(*p*,*s*)[2]
• _Decimal32	996/997	4	DECFLOAT(16)[7, 8]
• _Decimal64	996/997	8	DECFLOAT(16)[8]
• _Decimal128	996/997	16	DECFLOAT(34)[8]
float	480	4	FLOAT (single precision)
double	480	8	FLOAT (double precision)
• SQL TYPE IS BINARY(*n*), 1<=*n*<=255	912	*n*	BINARY(*n*)
• SQL TYPE IS VARBINARY(*n*), 1<=*n*<=32704	908	*n*	VARBINARY(*n*)
Single-character form	452	1	CHAR(1)
NUL-terminated character form	460	*n*	VARCHAR (*n*-1)

Table 109. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in C programs (continued)

C host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
VARCHAR structured form 1<=*n*<=255	448	*n*	VARCHAR(*n*)
VARCHAR structured form *n*>255	456	*n*	VARCHAR(*n*)
Single-graphic form	468	1	GRAPHIC(1)
NUL-terminated graphic form	400	*n*	VARGRAPHIC (*n*-1)
VARGRAPHIC structured form 1<=*n*<128	464	*n*	VARGRAPHIC(*n*)
VARGRAPHIC structured form *n*>127	472	*n*	VARGRAPHIC(*n*)
• SQL TYPE IS RESULT_SET _LOCATOR	972	4	Result set locator[3]
SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	976	4	Table locator[3]
SQL TYPE IS BLOB_LOCATOR	960	4	BLOB locator[3]
SQL TYPE IS CLOB_LOCATOR	964	4	CLOB locator[3]
SQL TYPE IS DBCLOB_LOCATOR	968	4	DBCLOB locator[3]
SQL TYPE IS BLOB(*n*) 1≤*n*≤2147483647	404	*n*	BLOB(*n*)
SQL TYPE IS CLOB(*n*) 1≤*n*≤2147483647	408	*n*	CLOB(*n*)
SQL TYPE IS DBCLOB(*n*) 1≤*n*≤1073741823	412	*n*	DBCLOB(*n*)[4]
SQL TYPE IS XML AS BLOB(*n*)	404	0	XML

Table 109. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in C programs (continued)

C host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
SQL TYPE IS XML AS CLOB(*n*)	408	0	XML
SQL TYPE IS XML AS DBCLOB(*n*)	412	0	XML
SQL TYPE IS BLOB_FILE	916/917	267	BLOB file reference [3]
SQL TYPE IS CLOB_FILE	920/921	267	CLOB file reference [3]
SQL TYPE IS DBCLOB_FILE	924/925	267	DBCLOB file reference [3]
SQL TYPE IS XML AS BLOB_FILE	916/917	267	XML BLOB file reference [3]
SQL TYPE IS XML AS CLOB_FILE	920/921	267	XML CLOB file reference [3]
SQL TYPE IS XML AS DBCLOB_FILE	924/925	267	XML DBCLOB file reference [3]
SQL TYPE IS ROWID	904	40	ROWID

Notes:

1. If a host variable includes an indicator variable, the SQLTYPE value is the base SQLTYPE value plus 1.
2. *p* is the *precision*; in SQL terminology, this the total number of digits. In C, this is called the *size*.

   s is the *scale*; in SQL terminology, this is the number of digits to the right of the decimal point. In C, this is called the *precision*.

   C++ does not support the decimal data type.
3. Do not use this data type as a column type.
4. *n* is the number of double-byte characters.
5. No exact equivalent. Use DECIMAL(19,0).
6. The C data type long maps to the SQL data type BIGINT.
7. DFP host variable with a length of 4 is supported while DFP column can be defined only with length 8(DECFLOAT(16)) or 16(DECFLOAT(34)).
8. To use the decimal floating-point host data type, you must do the following:
   - Use z/OS 1.10 or later (z/OS V1R10 XL C/C++).
   - Compile with the C/C++ compiler option, DFP.
   - Specify the SQL compiler option to enable the Db2 coprocessor.
   - Specify C/C++ compiler option, ARCH(7). It is required by the DFP compiler option if the DFP type is used in the source.
   - Specify 'DEFINE(__STDC_WANT_DEC_FP__)' compiler option because DFP is not officially part of the C/C++ Language Standard.

The following table shows equivalent C host variables for each SQL data type. Use this table to determine the C data type for host variables that you define to receive output from the database. For example, if you retrieve TIMESTAMP data, you can define a variable of NUL-terminated character form or VARCHAR structured form

This table shows direct conversions between SQL data types and C data types. However, a number of SQL data types are compatible. When you do assignments or comparisons of data that have compatible data types, Db2 converts those compatible data types.

Table 110. C host variable equivalents that you can use when retrieving data of a particular SQL data type

SQL data type	C host variable equivalent	Notes
SMALLINT	short int	
INTEGER	long int	
DECIMAL(*p,s*) **or** NUMERIC(*p,s*)	decimal	You can use the double data type if your C compiler does not have a decimal data type; however, double is not an exact equivalent.
REAL **or** FLOAT(*n*)	float	1<=*n*<=21
DOUBLE PRECISION **or** FLOAT(*n*)	double	22<=*n*<=53
DECFLOAT(16)	_Decminal32	
DECFLOAT(34)	_Decimal128	
BIGINT	long long, long long int, and sqlint64	
BINARY(*n*)	SQL TYPE IS BINARY(*n*)	1<=*n*<=255 If data can contain character NULs (\0), certain C and C++ library functions might not handle the data correctly. Ensure that your application handles the data properly.
VARBINARY(*n*)	SQL TYPE IS VARBINARY(*n*)	1<=*n*<=32704
CHAR(1)	single-character form	
CHAR(*n*)	no exact equivalent	If *n*>1, use NUL-terminated character form
VARCHAR(*n*)	NUL-terminated character form	If data can contain character NULs (\0), use VARCHAR structured form. Allow at least *n*+1 to accommodate the NUL-terminator.
	VARCHAR structured form	
GRAPHIC(1)	single-graphic form	
GRAPHIC(*n*)	no exact equivalent	If *n*>1, use NUL-terminated graphic form. *n* is the number of double-byte characters.
VARGRAPHIC(*n*)	NUL-terminated graphic form	If data can contain graphic NUL values (\0\0), use VARGRAPHIC structured form. Allow at least *n*+1 to accommodate the NUL-terminator. *n* is the number of double-byte characters.
	VARGRAPHIC structured form	*n* is the number of double-byte characters.

Table 110. C host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	C host variable equivalent	Notes
DATE	NUL-terminated character form	If you are using a date exit routine, that routine determines the length. Otherwise, allow at least 11 characters to accommodate the NUL-terminator.
	VARCHAR structured form	If you are using a date exit routine, that routine determines the length. Otherwise, allow at least 10 characters.
TIME	NUL-terminated character form	If you are using a time exit routine, the length is determined by that routine. Otherwise, the length must be at least 7; to include seconds, the length must be at least 9 to accommodate the NUL-terminator.
	VARCHAR structured form	If you are using a time exit routine, the length is determined by that routine. Otherwise, the length must be at least 6; to include seconds, the length must be at least 8.
TIMESTAMP	NUL-terminated character form	The length must be at least 20. To include microseconds, the length must be 27. If the length is less than 27, truncation occurs on the microseconds part.
	VARCHAR structured form	The length must be at least 19. To include microseconds, the length must be 26. If the length is less than 26, truncation occurs on the microseconds part.
TIMESTAMP(0)	NUL-terminated character form	The length must be at least 20.
	VARCHAR structured form	The length must be at least 19.
TIMESTAMP(*p*) *p* > 0	NUL-terminated character form	The length must be at least 20. To include fractional seconds, the length must be 21+*x* where *x* is the number of fractional seconds to include; if *x* is less than *p*, truncation occurs on the fraction seconds part.
	VARCHAR structured form	The length must be at least 19. To include fractional seconds, the length must be 20+*x* where *x* is the number of fractional seconds to include; if *x* is less than *p*, truncation occurs on the fractional seconds part.
TIMESTAMP(0) WITH TIME ZONE	NUL-terminated character form	The length must be at least 26.
	VARCHAR structured form	The length must be at least 25.

Table 110. C host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	C host variable equivalent	Notes
TIMESTAMP(*p*) WITH TIME ZONE	NUL-terminated character form	The length must be at least 27+*p*.
	VARCHAR structured form	The length must be at least 26+*p*.
Result set locator	SQL TYPE IS RESULT_SET_LOCATOR	Use this data type only for receiving result sets. Do not use this data type as a column type.
Table locator	SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. Do not use this data type as a column type.
BLOB locator	SQL TYPE IS BLOB_LOCATOR	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.
CLOB locator	SQL TYPE IS CLOB_LOCATOR	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB locator	SQL TYPE IS DBCLOB_LOCATOR	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.
BLOB(*n*)	SQL TYPE IS BLOB(*n*)	1≤*n*≤2147483647
CLOB(*n*)	SQL TYPE IS CLOB(*n*)	1≤*n*≤2147483647
DBCLOB(*n*)	SQL TYPE IS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤*n*≤1073741823
XML	SQL TYPE IS XML AS BLOB(*n*)	1≤*n*≤2147483647
XML	SQL TYPE IS XML AS CLOB(*n*)	1≤*n*≤2147483647
XML	SQL TYPE IS XML AS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤*n*≤1073741823
BLOB file reference	SQL TYPE IS BLOB_FILE	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.
CLOB file reference	SQL TYPE IS CLOB_FILE	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB file reference	SQL TYPE IS DBCLOB_FILE	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.
XML BLOB file reference	SQL TYPE IS XML AS BLOB_FILE	Use this data type only to manipulate XML data as BLOB files. Do not use this data type as a column type.
XML CLOB file reference	SQL TYPE IS XML AS CLOB_FILE	Use this data type only to manipulate XML data as CLOB files. Do not use this data type as a column type.

Table 110. C host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	C host variable equivalent	Notes
XML DBCLOB file reference	SQL TYPE IS XML AS DBCLOB_FILE	Use this data type only to manipulate XML data as DBCLOB files. Do not use this data type as a column type.
ROWID	SQL TYPE IS ROWID	

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

Host variable data types for XML data in embedded SQL applications (Db2 Programming for XML)

Handling SQL error codes in C and C++ applications

C and C++ applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

To request information about SQL errors in C and C++ applications, use the following approaches:

- You can use the subroutine DSNTIAR to convert an SQL return code into a text message.

  DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program. For concepts and more information about the behavior of DSNTIAR, see "Displaying SQLCA fields by calling DSNTIAR" on page 529.

  DSNITAR syntax

  DSNTIAR has the following syntax:

  ```
  rc = DSNTIAR(&sqlca, &message, &lrecl);
  ```

  Parameters for DSNTIAR

  The DSNTIAR parameters have the following meanings:

  &sqlca
  An SQL communication area.

  &message
  An output area, in VARCHAR format, in which DSNTIAR places the message text. The first halfword contains the length of the remaining area; its minimum value is 240.

  The output lines of text, each line being the length specified in *&lrecl*, are put into this area. For example, you could specify the format of the output area as:

  ```
  #define data_len 132
  #define data_dim 10
  int length_of_line = data_len ;
  struct error_struct {
     short int error_len;
     char error_text[data_dim][data_len];
     } error_message = {data_dim * data_len};
  ⋮
  rc = DSNTIAR(&sqlca, &error_message, &length_of_line);
  ```

where error_message is the name of the message output area, data_dim is the number of lines in the message output area, and data_len is the length of each line.

&lrecl
A fullword containing the logical record length of output messages, between 72 and 240.

To inform your compiler that DSNTIAR is an assembler language program, include one of the following statements in your application.

For C, include:

```
#pragma linkage (DSNTIAR,OS)
```

For C++, include a statement similar to this:

```
extern "OS" short int DSNTIAR(struct sqlca *sqlca,
                              struct error_struct *error_message,
                              int *data_len);
```

Examples of calling DSNTIAR from an application appear in the Db2 sample C program DSN8BD3 and in the sample C++ program DSN8BE3. Both are in the library DSN8B10.SDSNSAMP. See "Sample applications supplied with Db2 for z/OS" on page 1010 for instructions on how to access and print the source code for the sample programs.

- If your CICS application requires CICS storage handling, you must use the subroutine DSNTIAC instead of DSNTIAR.

DSNTIAC syntax
DSNTIAC has the following syntax:

```
rc = DSNTIAC(&eib, &commarea, &sqlca, &message, &lrecl);
```

Parameters for DSNTIAC

DSNTIAC has extra parameters, which you must use for calls to routines that use CICS commands.

&eib
EXEC interface block

&commarea
communication area

For more information on these parameters, see the appropriate application programming guide for CICS. The remaining parameter descriptions are the same as those for DSNTIAR. Both DSNTIAC and DSNTIAR format the SQLCA in the same way.

You must define DSNTIA1 in the CSD. If you load DSNTIAR or DSNTIAC, you must also define them in the CSD. For an example of CSD entry generation statements for use with DSNTIAC, see job DSNTEJ5A.

The assembler source code for DSNTIAC and job DSNTEJ5A, which assembles and link-edits DSNTIAC, are in the data set *prefix*.SDSNSAMP.

- You can also use the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement to convert an SQL return code into a text message.

  Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR. For more information about GET DIAGNOSTICS, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

Related tasks
Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference
GET DIAGNOSTICS (Db2 SQL)

COBOL applications that issue SQL statements

You can code SQL statements in certain COBOL program sections.

The allowable sections are shown in the following table.

Table 111. Allowable SQL statements for COBOL program sections

SQL statement	Program section
BEGIN DECLARE SECTION END DECLARE SECTION	WORKING-STORAGE SECTION[1] **or** LINKAGE SECTION
INCLUDE SQLCA	WORKING-STORAGE SECTION[1] **or** LINKAGE SECTION
INCLUDE text-file-name	PROCEDURE DIVISION **or** DATA DIVISION[2]
DECLARE TABLE DECLARE CURSOR	DATA DIVISION **or** PROCEDURE DIVISION
DECLARE VARIABLE	WORKING-STORAGE SECTION[1]
Other	PROCEDURE DIVISION

Notes:

1. If you use the Db2 coprocessor, you can use the LOCAL-STORAGE SECTION wherever WORKING-STORAGE SECTION is listed in the table.
2. When including host variable declarations, the INCLUDE statement must be in the WORKING-STORAGE SECTION or the LINKAGE SECTION.

You cannot put SQL statements in the DECLARATIVES section of a COBOL program.

Each SQL statement in a COBOL program must begin with EXEC SQL and end with END-EXEC. If you are using the Db2 precompiler, the EXEC and SQL keywords must appear on one line, but the remainder of the statement can appear on subsequent lines. If you are using the Db2 coprocessor, the EXEC and SQL keywords can be on different lines. Do not include any tokens between the two keywords EXEC and SQL except for COBOL comments, including debugging lines. Do not include SQL comments between the keywords EXEC and SQL.

If the SQL statement appears between two COBOL statements, the period after END-EXEC is optional and might not be appropriate. If the statement appears in an IF...THEN set of COBOL statements, omit the ending period to avoid inadvertently ending the IF statement.

You might code an UPDATE statement in a COBOL program as follows:

```
EXEC SQL
    UPDATE DSN8B10.DEPT
    SET MGRNO = :MGR-NUM
    WHERE DEPTNO = :INT-DEPT
END-EXEC.
```

Comments

You can include COBOL comment lines (* in column 7) in SQL statements wherever you can use a blank.

Also, you can include SQL comments (' --') in any embedded SQL statement. A space must precede the two hyphens (' --') that begin the comment. For more information, see SQL comments (Db2 SQL).

Restrictions: If you are using the Db2 precompiler, be aware of the following restrictions:

- You cannot include COBOL comment lines between the keywords EXEC and SQL. The precompiler treats COBOL debugging lines and page-eject lines (/ in column 7) as comment lines. The Db2 coprocessor treats the debugging lines based on the COBOL rules, which depend on the WITH DEBUGGING mode setting.
- You cannot use COBOL inline comments that are identified by a floating comment indicator (*>). COBOL inline comments are interpreted correctly only when the Db2 coprocessor is used.

For an SQL INCLUDE statement, the Db2 precompiler treats any text that follows the period after END-EXEC, and on the same line as END-EXEC, as a comment. The Db2 coprocessor treats this text as part of the COBOL program syntax.

Debugging lines

The Db2 precompiler ignores the 'D' in column 7 on debugging lines and treats it as a blank. The Db2 coprocessor follows the COBOL language rules regarding debugging lines.

Continuation for SQL statements

The rules for continuing a character string constant from one line to the next in an SQL statement embedded in a COBOL program are the same as those for continuing a non-numeric literal in COBOL. However, you can use either a quote or an apostrophe as the first nonblank character in area B of the continuation line. The same rule applies for the continuation of delimited identifiers and does not depend on the string delimiter option.

To conform with SQL standard, delimit a character string constant with an apostrophe, and use a quote as the first nonblank character in area B of the continuation line for a character string constant.

Continued lines of an SQL statement can be in columns 8 through 72 when using the Db2 precompiler and columns 12 through 72 when using the Db2 coprocessor.

Delimiters

Delimit an SQL statement in your COBOL program with the beginning keyword EXEC SQL and an `END-EXEC` as shown in the following example code:

```
EXEC SQL
  SQL-statement
END-EXEC.
```

COPY

If you use the Db2 precompiler, do not use a COBOL COPY statement within host variable declarations. If you use the Db2 coprocessor, you can use COBOL COPY.

REPLACE

If you use the Db2 precompiler, the REPLACE statement has no effect on SQL statements. It affects only the COBOL statements that the precompiler generates.

If you use the Db2 coprocessor, the REPLACE statement replaces text strings in SQL statements as well as in generated COBOL statements.

Declaring tables and views

Your COBOL program should include the statement DECLARE TABLE to describe each table and view the program accesses. You can use the Db2 declarations generator (DCLGEN) to generate the DECLARE TABLE statements. You should include the DCLGEN members in the DATA DIVISION.

Dynamic SQL in a COBOL program

In general, COBOL programs can easily handle dynamic SQL statements. COBOL programs can handle SELECT statements if the data types and the number of fields returned are fixed. If you want to use variable-list SELECT statements, use an SQLDA.

Including code

To include SQL statements or COBOL host variable declarations from a member of a partitioned data set, use the following SQL statement in the source code where you want to include the statements:

```
EXEC SQL INCLUDE member-name END-EXEC.
```

If you are using the Db2 precompiler, you cannot nest SQL INCLUDE statements. In this case, do not use COBOL verbs to include SQL statements or host variable declarations, and do not use the SQL

INCLUDE statement to include CICS preprocessor related code. In general, if you are using the Db2 precompiler, use the SQL INCLUDE statement only for SQL-related coding. If you are using the COBOL Db2 coprocessor, none of these restrictions apply.

Use the 'EXEC SQL' and 'END-EXEC' keyword pair to include SQL statements only. COBOL statements, such as COPY or REPLACE, are not allowed.

Margins
You must code SQL statements that begin with EXEC SQL in columns 12 through 72. Otherwise the Db2 precompiler does not recognize the SQL statement.

Names
You can use any valid COBOL name for a host variable. Do not use external entry names or access plan names that begin with 'DSN', and do not use host variable names that begin with 'SQL'. These names are reserved for Db2.

Sequence numbers
The source statements that the Db2 precompiler generates do not include sequence numbers.

Statement labels
You can precede executable SQL statements in the PROCEDURE DIVISION with a paragraph name.

WHENEVER statement
The target for the GOTO clause in an SQL statement WHENEVER must be a section name or unqualified paragraph name in the PROCEDURE DIVISION.

Special COBOL considerations
The following considerations apply to programs written in COBOL:

- In a COBOL program that uses elements in a multi-level structure as host variable names, the Db2 precompiler generates the lowest two-level names.
- Using the COBOL compiler options DYNAM and NODYNAM depends on the operating environment.

  TSO and IMS: You can specify the option DYNAM when compiling a COBOL program if you use the following guidelines. IMS and Db2 share a common alias name, DSNHLI, for the language interface module. You must do the following when you concatenate your libraries:

  – If you use IMS with the COBOL option DYNAM, be sure to concatenate the IMS library first.
  – If you run your application program only under , be sure to concatenate the Db2 library first.

  CICS, CAF, and RRSAF: You must specify the NODYNAM option when you compile a COBOL program that either includes CICS statements or is translated by a separate CICS translator or the integrated CICS translator. In these cases, you cannot specify the DYNAM option. If your CICS program has a subroutine that is not translated by a separate CICS translator or the integrated CICS translator but contains SQL statements, you can specify the DYNAM option. However, in this case, you must concatenate the CICS libraries before the Db2 libraries.

  You can compile COBOL stored procedures with either the DYNAM option or the NODYNAM option. If you use DYNAM, ensure that the correct Db2 language interface module is loaded dynamically by performing one of the following actions:

  – Use the ATTACH(RRSAF) precompiler option.
  – Copy the DSNRLI module into a load library that is concatenated in front of the Db2 libraries. Use the member name DSNHLI.
- To avoid truncating numeric values, use either of the following methods:

  – Use the COMP-5 data type for binary integer host variables.
  – Specify the COBOL compiler option:

    - TRUNC(OPT) if you are certain that the data being moved to each binary variable by the application does not have a larger precision than is defined in the PICTURE clause of the binary variable.
    - TRUNC(BIN) if the precision of data being moved to each binary variable might exceed the value in the PICTURE clause.

Db2 assigns values to binary integer host variables as if you had specified the COBOL compiler option TRUNC(BIN) or used the COMP-5 data type.

- If you are using the Db2 precompiler and your COBOL program contains several entry points or is called several times, the USING clause of the entry statement that executes before the first SQL statement executes must contain the SQLCA and all linkage section entries that any SQL statement uses as host variables.
- If you use the Db2 precompiler, no compiler directives should appear between the PROCEDURE DIVISION and the DECLARATIVES statement.
- Do not use COBOL figurative constants (such as ZERO and SPACE), symbolic characters, reference modification, and subscripts within SQL statements.
- Observe the rules for naming SQL identifiers, as described in SQL identifiers (Db2 SQL). However, for COBOL only, the names of SQL identifiers can follow the rules for naming COBOL words, as described in COBOL words with single-byte characters (COBOL) (Enterprise COBOL for z/OS Programming Guide). However, the names must not exceed the allowable length for the Db2 object.
- Observe these rules for hyphens:
  - Surround hyphens used as subtraction operators with spaces. Db2 usually interprets a hyphen with no spaces around it as part of a host variable name.
  - You can use hyphens in SQL identifiers under either of the following circumstances:
    - The application program is a local application that runs on Db2 for z/OS Db2 9 or later.
    - The application program accesses remote sites, and the local site and remote sites are Db2 for z/OS Db2 9 or later.
- If you include an SQL statement in a COBOL `PERFORM … THRU` paragraph and also specify the SQL statement `WHENEVER … GO`, the COBOL compiler returns the warning message IGYOP3094. That message might indicate a problem. This usage is not recommended.
- If you are using the Db2 precompiler, all SQL statements and any host variables they reference must be within the first program when using nested programs or batch compilation.
- If you are using the Db2 precompiler, your COBOL programs must have a DATA DIVISION and a PROCEDURE DIVISION. Both divisions and the WORKING-STORAGE SECTION must be present in programs that contain SQL statements. However, if your COBOL programs requires the LOCAL-STORAGE SECTION, then the Db2 coprocessor should be used instead of the Db2 precompiler.
- The Db2 precompiler generates this COBOL variable:

  ```
  DSN-TMP2 PIC S9(18) COMP-3
  ```

  The Db2 coprocessor generates this COBOL variable:

  ```
  SQL---SCRVALD DS 10P PIC S9(18) COMP-3
  ```

  If you specify COBOL option RULES(NOEVENPACK), the COBOL compiler generates warning IGYDS1348-W, because those variables have an even number of packed decimal digits.

PSPI If your program uses the Db2 precompiler and uses parameters that are defined in LINKAGE SECTION as host variables to Db2 and the address of the input parameter might change on subsequent invocations of your program, your program must reset the variable SQL-INIT-FLAG. This flag is generated by the Db2 precompiler. Resetting this flag indicates that the storage must initialize when the next SQL statement executes. To reset the flag, insert the statement MOVE ZERO TO SQL-INIT-FLAG in the called program's PROCEDURE DIVISION, ahead of any executable SQL statements that use the host variables. If you use the COBOL Db2 coprocessor, the called program does not need to reset SQL-INIT-FLAG. PSPI

Handling SQL error codes

Cobol applications can request more information about SQL errors from Db2. For more information, see “Handling SQL error codes in Cobol applications” on page 683.

Related concepts

Sample applications supplied with Db2 for z/OS
Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

DCLGEN (declarations generator)
Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

SQL identifiers (Db2 SQL)

Related tasks

Overview of programming applications that access Db2 for z/OS data
Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Checking the execution of SQL statements by using the GET DIAGNOSTICS statement
One way to check whether an SQL statement executed successfully is to ask Db2 to return the diagnostic information about the last executed SQL statement.

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Displaying SQLCA fields by calling DSNTIAR
If you use the SQLCA to check whether an SQL statement executed successfully, your program needs to read the data in the appropriate SQLCA fields. One easy way to read these fields is to use the assembler subroutine DSNTIAR.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

COBOL programming examples

You can write Db2 programs in COBOL. These programs can access a local or remote Db2 subsystem and can execute static or dynamic SQL statements. This information contains several such programming examples.

To prepare and run these applications, use the JCL in *prefix*.SDSNSAMP as a model for your JCL.

Related reference

Assembler, C, C++, COBOL, PL/I, and REXX programming examples (Db2 Programming samples)

Sample COBOL dynamic SQL program

You can code dynamic varying-list SELECT statements in a COBOL program. *Varying-List SELECT statements* are statements for which you do not know the number or data types of columns that are to be returned when you write the program.

Introductory concepts

Submitting SQL statements to Db2 (Introduction to Db2 for z/OS)
Dynamic SQL applications (Introduction to Db2 for z/OS)

“Including dynamic SQL in your program” on page 496 describes three variations of dynamic SQL statements:

- Non-SELECT statements
- Fixed-List SELECT statements

  In this case, you know the number of columns returned and their data types when you write the program.
- Varying-List SELECT statements.

  In this case, you do **not** know the number of columns returned and their data types when you write the program.

This section documents a technique of coding varying list SELECT statements in COBOL.

This example program does not support BLOB, CLOB, or DBCLOB data types.

Pointers and based variables in the sample COBOL program

COBOL has a POINTER type and a SET statement that provide pointers and based variables.

The SET statement sets a pointer from the address of an area in the linkage section or another pointer; the statement can also set the address of an area in the linkage section. UNLDBCU2 in “Example of the sample COBOL program” on page 624 provides these uses of the SET statement. The SET statement does not permit the use of an address in the WORKING-STORAGE section.

Storage allocation for the sample COBOL program

COBOL does not provide a means to allocate main storage within a program. You can achieve the same end by having an initial program which allocates the storage, and then calls a second program that manipulates the pointer. (COBOL does not permit you to directly manipulate the pointer because errors and abends are likely to occur.)

The initial program is extremely simple. It includes a working storage section that allocates the maximum amount of storage needed. This program then calls the second program, passing the area or areas on the CALL statement. The second program defines the area in the linkage section and can then use pointers within the area.

If you need to allocate parts of storage, the best method is to use indexes or subscripts. You can use subscripts for arithmetic and comparison operations.

Example of the sample COBOL program

The following example shows an example of the initial program UNLDBCU1 that allocates the storage and calls the second program UNLDBCU2. UNLDBCU2 then defines the passed storage areas in its linkage section and includes the USING clause on its PROCEDURE DIVISION statement.

Defining the pointers, then redefining them as numeric, permits some manipulation of the pointers that you cannot perform directly. For example, you cannot add the column length to the record pointer, but you can add the column length to the numeric value that redefines the pointer.

The following example is the initial program that allocates storage.

```
**** UNLDBCU1- DB2 SAMPLE BATCH COBOL UNLOAD PROGRAM  ***********
*                                                               *
*   MODULE NAME = UNLDBCU1                                      *
*                                                               *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                  *
*                       UNLOAD PROGRAM                          *
*                       BATCH                                   *
*                       IBM ENTERPRISE COBOL FOR Z/OS           *
*                                                               *
*   COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987      *
*    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083      *
*                                                               *
*   STATUS = VERSION 1 RELEASE 3, LEVEL 0                       *
```

```
*                                                               *
*   FUNCTION = THIS MODULE PROVIDES THE STORAGE NEEDED BY       *
*              UNLDBCU2 AND CALLS THAT PROGRAM.                 *
*                                                               *
*   NOTES =                                                     *
*     DEPENDENCIES = ENTERPRISE COBOL FOR Z/OS IS REQUIRED.     *
*                    SEVERAL NEW FACILITIES ARE USED.           *
*                                                               *
*     RESTRICTIONS =                                            *
*               THE MAXIMUM NUMBER OF COLUMNS IS 750,           *
*               WHICH IS THE SQL LIMIT.                         *
*                                                               *
*               DATA RECORDS ARE LIMITED TO 32700 BYTES,        *
*               INCLUDING DATA, LENGTHS FOR VARCHAR DATA,       *
*               AND SPACE FOR NULL INDICATORS.                  *
*                                                               *
*   MODULE TYPE = IBM ENTERPRISE COBOL PROGRAM                  *
*      PROCESSOR   = ENTERPRISE COBOL FOR Z/OS                  *
*      MODULE SIZE = SEE LINK EDIT                              *
*      ATTRIBUTES  = REENTRANT                                  *
*                                                               *
*   ENTRY POINT = UNLDBCU1                                      *
*      PURPOSE = SEE FUNCTION                                   *
*      LINKAGE = INVOKED FROM DSN RUN                           *
*      INPUT   = NONE                                           *
*      OUTPUT  = NONE                                           *
*                                                               *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION               *
*                                                               *
*   EXIT-ERROR =                                                *
*      RETURN CODE = NONE                                       *
*      ABEND CODES =  NONE                                      *
*      ERROR-MESSAGES = NONE                                    *
*                                                               *
*   EXTERNAL REFERENCES =                                       *
*      ROUTINES/SERVICES =                                      *
*            UNLDBCU2 - ACTUAL UNLOAD PROGRAM                   *
*                                                               *
*      DATA-AREAS        =    NONE                              *
*      CONTROL-BLOCKS    =    NONE                              *
*                                                               *
*   TABLES = NONE                                               *
*   CHANGE-ACTIVITY = NONE                                      *
*                                                               *
*  *PSEUDOCODE*                                                 *
*                                                               *
*    PROCEDURE                                                  *
*    CALL UNLDBCU2.                                             *
*    END.                                                       *
*---------------------------------------------------------------*
/
 IDENTIFICATION DIVISION.
*-----------------------
 PROGRAM-ID.    UNLDBCU1
*
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
*
 01  WORKAREA-IND.
         02  WORKIND PIC S9(4) COMP OCCURS 750 TIMES.
 01  RECWORK.
         02  RECWORK-LEN PIC S9(8) COMP VALUE 32700.
         02  RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.
*
 PROCEDURE DIVISION.
*
          CALL 'UNLDBCU2' USING WORKAREA-IND RECWORK.
          GOBACK.
```

The following example is the called program that does pointer manipulation.

```
**** UNLDBCU2- DB2 SAMPLE BATCH COBOL UNLOAD PROGRAM  ***********
*                                                               *
*   MODULE NAME = UNLDBCU2                                      *
*                                                               *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                  *
```

```
      *                      UNLOAD PROGRAM                          *
      *                      BATCH                                   *
      *                      ENTERPRISE COBOL FOR Z/OS               *
      *                                                              *
      *   COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987     *
      *    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083     *
      *                                                              *
      *   STATUS = VERSION 1 RELEASE 3, LEVEL 0                      *
      *                                                              *
      *   FUNCTION = THIS MODULE ACCEPTS A TABLE NAME OR VIEW NAME   *
      *              AND UNLOADS THE DATA IN THAT TABLE OR VIEW.     *
      *    READ IN A TABLE NAME FROM SYSIN.                          *
      *    PUT DATA FROM THE TABLE INTO DD SYSREC01.                 *
      *    WRITE RESULTS TO SYSPRINT.                                *
      *                                                              *
      *   NOTES =                                                    *
      *     DEPENDENCIES = IBM ENTERPRISE COBOL FOR Z/OS             *
      *                    IS REQUIRED.                              *
      *                                                              *
      *     RESTRICTIONS =                                           *
      *               THE SQLDA IS LIMITED TO 33016 BYTES.           *
      *               THIS SIZE ALLOWS FOR THE DB2 MAXIMUM           *
      *               OF 750 COLUMNS.                                *
      *                                                              *
      *               DATA RECORDS ARE LIMITED TO 32700 BYTES,       *
      *               INCLUDING DATA, LENGTHS FOR VARCHAR DATA,      *
      *               AND SPACE FOR NULL INDICATORS.                 *
      *                                                              *
      *               TABLE OR VIEW NAMES ARE ACCEPTED, AND ONLY     *
      *               ONE NAME IS ALLOWED PER RUN.                   *
      *                                                              *
      *   MODULE TYPE = ENTERPRISE COBOL FOR Z/OS                    *
      *      PROCESSOR   = DB2  PRECOMPILER, COBOL COMPILER          *
      *      MODULE SIZE = SEE LINK EDIT                             *
      *      ATTRIBUTES  = REENTRANT                                 *
      *                                                              *
      *   ENTRY POINT = UNLDBCU2                                     *
      *      PURPOSE = SEE FUNCTION                                  *
      *      LINKAGE =                                               *
      *         CALL 'UNLDBCU2' USING WORKAREA-IND RECWORK.          *
      *                                                              *
      *      INPUT   = SYMBOLIC LABEL/NAME = WORKAREA-IND            *
      *                DESCRIPTION = INDICATOR VARIABLE ARRAY        *
      *                01  WORKAREA-IND.                             *
      *                  02  WORKIND PIC S9(4) COMP OCCURS 750 TIMES. *
      *                                                              *
      *                SYMBOLIC LABEL/NAME = RECWORK                 *
      *                DESCRIPTION = WORK AREA FOR OUTPUT RECORD     *
      *                01  RECWORK.                                  *
      *                  02  RECWORK-LEN PIC S9(8) COMP.             *
      *                  02  RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.*
      *                                                              *
      *                SYMBOLIC LABEL/NAME = SYSIN                   *
      *                DESCRIPTION = INPUT REQUESTS - TABLE OR VIEW  *
      *                                                              *
      *      OUTPUT  = SYMBOLIC LABEL/NAME = SYSPRINT                *
      *                DESCRIPTION = PRINTED RESULTS                 *
      *                                                              *
      *                SYMBOLIC LABEL/NAME = SYSREC01                *
      *                DESCRIPTION = UNLOADED TABLE DATA             *
      *                                                              *
      *   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION              *
      *   EXIT-ERROR =                                               *
      *      RETURN CODE = NONE                                      *
      *      ABEND CODES =  NONE                                     *
      *      ERROR-MESSAGES =                                        *
      *          DSNT490I SAMPLE COBOL DATA UNLOAD PROGRAM RELEASE 3.0*
      *                   -  THIS IS THE HEADER, INDICATING A NORMAL  *
      *                   -  START FOR THIS PROGRAM.                 *
      *          DSNT493I SQL ERROR, SQLCODE = NNNNNNNN              *
      *                   -  AN SQL ERROR OR WARNING WAS ENCOUNTERED  *
      *                   -  ADDITIONAL INFORMATION FROM DSNTIAR     *
      *                   -  FOLLOWS THIS MESSAGE.                   *
      *          DSNT495I SUCCESSFUL UNLOAD XXXXXXXX ROWS OF         *
      *                   TABLE TTTTTTTT                             *
      *                   -  THE UNLOAD WAS SUCCESSFUL.  XXXXXXXX IS  *
      *                   -  THE NUMBER OF ROWS UNLOADED.  TTTTTTTT   *
      *                   -  IS THE NAME OF THE TABLE OR VIEW FROM   *
      *                   -  WHICH IT WAS UNLOADED.                  *
      *          DSNT496I UNRECOGNIZED DATA TYPE CODE OF NNNNN       *
      *                   -  THE PREPARE RETURNED AN INVALID DATA    *
      *                   -  TYPE CODE.  NNNNN IS THE CODE, PRINTED   *
```

```
*                   -  IN DECIMAL.  USUALLY AN ERROR IN         *
*                   -  THIS ROUTINE OR A NEW DATA TYPE.         *
*          DSNT497I RETURN CODE FROM MESSAGE ROUTINE DSNTIAR    *
*                   -  THE MESSAGE FORMATTING ROUTINE DETECTED  *
*                   -  AN ERROR.  SEE THAT ROUTINE FOR RETURN   *
*                   -  CODE INFORMATION.  USUALLY AN ERROR IN   *
*                   -  THIS ROUTINE.                            *
*          DSNT498I ERROR, NO VALID COLUMNS FOUND               *
*                   -  THE PREPARE RETURNED DATA WHICH DID NOT  *
*                   -  PRODUCE A VALID OUTPUT RECORD.           *
*                   -  USUALLY AN ERROR IN THIS ROUTINE.        *
*          DSNT499I NO ROWS FOUND IN TABLE OR VIEW              *
*                   -  THE CHOSEN TABLE OR VIEWS DID NOT        *
*                   -  RETURN ANY ROWS.                         *
*          ERROR MESSAGES FROM MODULE DSNTIAR                   *
*                   -  WHEN AN ERROR OCCURS, THIS MODULE        *
*                   -  PRODUCES CORRESPONDING MESSAGES.         *
*          OTHER MESSAGES:                                      *
*               THE TABLE COULD NOT BE UNLOADED. EXITING.       *
*                                                               *
*   EXTERNAL REFERENCES =                                       *
*      ROUTINES/SERVICES =                                      *
*            DSNTIAR  - TRANSLATE SQLCA INTO MESSAGES           *
*      DATA-AREAS        =    NONE                              *
*      CONTROL-BLOCKS    =                                      *
*            SQLCA    - SQL COMMUNICATION AREA                  *
*                                                               *
*   TABLES = NONE                                               *
*   CHANGE-ACTIVITY = NONE                                      *
*                                                               *
*  *PSEUDOCODE*                                                 *
*    PROCEDURE                                                  *
*    EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.               *
*    EXEC SQL DECLARE SEL STATEMENT END-EXEC.                   *
*    INITIALIZE THE DATA, OPEN FILES.                           *
*    OBTAIN STORAGE FOR THE SQLDA AND THE DATA RECORDS.         *
*    READ A TABLE NAME.                                         *
*    OPEN SYSREC01.                                             *
*    BUILD THE SQL STATEMENT TO BE EXECUTED                     *
*    EXEC SQL PREPARE SQL STATEMENT INTO SQLDA END-EXEC.        *
*    SET UP ADDRESSES IN THE SQLDA FOR DATA.                    *
*    INITIALIZE DATA RECORD COUNTER TO 0.                       *
*    EXEC SQL OPEN DT END-EXEC.                                 *
*    DO WHILE SQLCODE IS 0.                                     *
*    EXEC SQL FETCH DT USING DESCRIPTOR SQLDA END-EXEC.         *
*    ADD IN MARKERS TO DENOTE NULLS.                            *
*    WRITE THE DATA TO SYSREC01.                                *
*    INCREMENT DATA RECORD COUNTER.                             *
*    END.                                                       *
*    EXEC SQL CLOSE DT END-EXEC.                                *
*    INDICATE THE RESULTS OF THE UNLOAD OPERATION.              *
*    CLOSE THE SYSIN, SYSPRINT, AND SYSREC01 FILES.             *
*    END.                                                       *
*---------------------------------------------------------------*
/
 IDENTIFICATION DIVISION.
*-----------------------
 PROGRAM-ID.    UNLDBCU2
*
 ENVIRONMENT DIVISION.
*--------------------
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT SYSIN
            ASSIGN TO DA-S-SYSIN.
     SELECT SYSPRINT
            ASSIGN TO UT-S-SYSPRINT.
     SELECT SYSREC01
            ASSIGN TO DA-S-SYSREC01.
*
 DATA DIVISION.
*-------------
*
 FILE SECTION.
 FD      SYSIN
         RECORD CONTAINS 80 CHARACTERS
         BLOCK CONTAINS 0 RECORDS
         LABEL RECORDS ARE OMITTED
         RECORDING MODE IS F.
 01  CARDREC                    PIC X(80).
*
```

```
 FD  SYSPRINT
         RECORD CONTAINS 120 CHARACTERS
         LABEL RECORDS ARE OMITTED
         DATA RECORD IS MSGREC
         RECORDING MODE IS F.
 01  MSGREC                     PIC X(120).
*
 FD  SYSREC01
         RECORD CONTAINS 5 TO 32704 CHARACTERS
         LABEL RECORDS ARE OMITTED
         DATA RECORD IS REC01
         RECORDING MODE IS V.
 01  REC01.
          02  REC01-LEN PIC S9(8) COMP.
          02  REC01-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
                          DEPENDING ON REC01-LEN.
/
 WORKING-STORAGE SECTION.
*
*****************************************************
* STRUCTURE FOR INPUT                               *
*****************************************************
 01  IOAREA.
         02  TNAME         PIC X(72).
         02  FILLER        PIC X(08).
 01  STMTBUF.
         49  STMTLEN       PIC S9(4) COMP VALUE 92.
         49  STMTCHAR      PIC X(92).
 01  STMTBLD.
         02  FILLER        PIC X(20) VALUE 'SELECT * FROM'.
         02  STMTTAB       PIC X(72).
*
*****************************************************
* REPORT HEADER STRUCTURE                           *
*****************************************************
 01  HEADER.
         02  FILLER PIC X(35)
             VALUE ' DSNT490I SAMPLE COBOL DATA UNLOAD '.
         02  FILLER PIC X(85) VALUE 'PROGRAM RELEASE 3.0'.
 01  MSG-SQLERR.
         02  FILLER PIC X(31)
             VALUE ' DSNT493I SQL ERROR, SQLCODE = '.
         02  MSG-MINUS       PIC X(1).
         02  MSG-PRINT-CODE  PIC 9(8).
         02  FILLER PIC X(81) VALUE '            '.
 01  MSG-OTHER-ERR.
         02  FILLER PIC X(42)
             VALUE ' THE TABLE COULD NOT BE UNLOADED. EXITING.'.
         02  FILLER PIC X(78) VALUE '            '.
 01  UNLOADED.
         02  FILLER PIC X(28)
             VALUE ' DSNT495I SUCCESSFUL UNLOAD '.
         02  ROWS   PIC 9(8).
         02  FILLER PIC X(15) VALUE ' ROWS OF TABLE '.
         02  TABLENAM PIC X(72) VALUE '        '.
 01  BADTYPE.
         02  FILLER PIC X(42)
             VALUE ' DSNT496I UNRECOGNIZED DATA TYPE CODE OF  '.
         02  TYPCOD PIC 9(8).
         02  FILLER PIC X(71) VALUE '         '.
 01  MSGRETCD.
         02  FILLER PIC X(42)
             VALUE ' DSNT497I RETURN CODE FROM MESSAGE ROUTINE'.
         02  FILLER PIC X(9) VALUE 'DSNTIAR '.
         02  RETCODE    PIC 9(8).
         02  FILLER PIC X(62) VALUE '          '.
 01  MSGNOCOL.
         02  FILLER PIC X(120)
             VALUE ' DSNT498I ERROR, NO VALID COLUMNS FOUND'.
 01  MSG-NOROW.
         02  FILLER PIC X(120)
             VALUE ' DSNT499I NO ROWS FOUND IN TABLE OR VIEW'.
*****************************************************
* WORKAREAS                                         *
*****************************************************
 77  NOT-FOUND         PIC S9(8) COMP VALUE  +100.
*****************************************************
* VARIABLES FOR ERROR-MESSAGE FORMATTING             *
*****************************************************
 01  ERROR-MESSAGE.
         02  ERROR-LEN   PIC S9(4)  COMP VALUE +960.
         02  ERROR-TEXT  PIC X(120) OCCURS 8 TIMES
```

```
                                    INDEXED BY ERROR-INDEX.
 77  ERROR-TEXT-LEN      PIC S9(8)  COMP VALUE +120.
*****************************************************
* SQL DESCRIPTOR AREA                               *
*****************************************************
 01  SQLDA.
         02  SQLDAID     PIC X(8)   VALUE 'SQLDA   '.
         02  SQLDABC     PIC S9(8) COMPUTATIONAL  VALUE 33016.
         02  SQLN        PIC S9(4) COMPUTATIONAL  VALUE 750.
         02  SQLD        PIC S9(4) COMPUTATIONAL  VALUE 0.
         02  SQLVAR      OCCURS 1 TO 750 TIMES
                                  DEPENDING ON SQLN.
             03  SQLTYPE     PIC S9(4) COMPUTATIONAL.
             03  SQLLEN      PIC S9(4) COMPUTATIONAL.
             03  SQLDATA     POINTER.
             03  SQLIND      POINTER.
             03  SQLNAME.
                 49  SQLNAMEL    PIC S9(4) COMPUTATIONAL.
                 49  SQLNAMEC    PIC X(30).
*
*  DATA TYPES FOUND IN SQLTYPE, AFTER REMOVING THE NULL BIT
*
 77  VARCTYPE            PIC S9(4)  COMP VALUE +448.
 77  CHARTYPE            PIC S9(4)  COMP VALUE +452.
 77  VARLTYPE            PIC S9(4)  COMP VALUE +456.
 77  VARGTYPE            PIC S9(4)  COMP VALUE +464.
 77  GTYPE               PIC S9(4)  COMP VALUE +468.
 77  LVARGTYP            PIC S9(4)  COMP VALUE +472.
 77  FLOATYPE            PIC S9(4)  COMP VALUE +480.
 77  DECTYPE             PIC S9(4)  COMP VALUE +484.
 77  INTTYPE             PIC S9(4)  COMP VALUE +496.
 77  HWTYPE              PIC S9(4)  COMP VALUE +500.
 77  DATETYP             PIC S9(4)  COMP VALUE +384.
 77  TIMETYP             PIC S9(4)  COMP VALUE +388.
 77  TIMESTMP            PIC S9(4)  COMP VALUE +392.
*
 01  RECPTR POINTER.
 01  RECNUM REDEFINES RECPTR PICTURE S9(8) COMPUTATIONAL.
 01  IRECPTR POINTER.
 01  IRECNUM REDEFINES IRECPTR PICTURE S9(8) COMPUTATIONAL.
 01  I      PICTURE S9(4) COMPUTATIONAL.
 01  J      PICTURE S9(4) COMPUTATIONAL.
 01  DUMMY  PICTURE S9(4) COMPUTATIONAL.
 01  MYTYPE PICTURE S9(4) COMPUTATIONAL.
 01  COLUMN-IND PICTURE S9(4) COMPUTATIONAL.
 01  COLUMN-LEN PICTURE S9(4) COMPUTATIONAL.
 01  COLUMN-PREC PICTURE S9(4) COMPUTATIONAL.
 01  COLUMN-SCALE PICTURE S9(4) COMPUTATIONAL.
 01  INDCOUNT           PIC S9(4) COMPUTATIONAL.
 01  ROWCOUNT           PIC S9(4) COMPUTATIONAL.
 01  ERR-FOUND PICTURE X(1).
 01  WORKAREA2.
         02  WORKINDPTR  POINTER    OCCURS 750 TIMES.
*****************************************************
*   DECLARE CURSOR AND STATEMENT FOR DYNAMIC SQL
*****************************************************
*
          EXEC SQL DECLARE DT CURSOR FOR SEL  END-EXEC.
          EXEC SQL DECLARE SEL STATEMENT      END-EXEC.
*
*****************************************************
* SQL INCLUDE FOR SQLCA                             *
*****************************************************
          EXEC SQL INCLUDE SQLCA  END-EXEC.
*
 77  ONE                 PIC S9(4)  COMP VALUE +1.
 77  TWO                 PIC S9(4)  COMP VALUE +2.
 77  FOUR                PIC S9(4)  COMP VALUE +4.
 77  QMARK               PIC X(1)        VALUE '?'.
*
 LINKAGE SECTION.
 01  LINKAREA-IND.
         02  IND   PIC   S9(4) COMP OCCURS 750 TIMES.
 01  LINKAREA-REC.
          02  REC1-LEN PIC S9(8) COMP.
          02  REC1-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
                             DEPENDING ON REC1-LEN.
 01  LINKAREA-QMARK.
         02  INDREC PIC   X(1).
/
 PROCEDURE DIVISION USING LINKAREA-IND LINKAREA-REC.
*
```

```
*****************************************************
* SQL RETURN CODE HANDLING                          *
*****************************************************
     EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER NOT FOUND  CONTINUE     END-EXEC.
*
*****************************************************
* MAIN PROGRAM ROUTINE                              *
*****************************************************
          SET IRECPTR TO ADDRESS OF REC1-CHAR(1).
*                                          **OPEN FILES
          MOVE 'N' TO ERR-FOUND.
*                                          **INITIALIZE
*                                          ** ERROR FLAG
          OPEN INPUT  SYSIN

               OUTPUT SYSPRINT
               OUTPUT SYSREC01.
*                                          **WRITE HEADER
          WRITE MSGREC FROM HEADER
                AFTER ADVANCING 2 LINES.
*                                          **GET FIRST INPUT
          READ SYSIN  RECORD INTO IOAREA.
*                                          **MAIN ROUTINE
          PERFORM PROCESS-INPUT THROUGH IND-RESULT.
*
 PROG-END.
*                                           **CLOSE FILES
          CLOSE SYSIN
                SYSPRINT
                SYSREC01.
          GOBACK.
/
***************************************************************
*                                                             *
*    PERFORMED SECTION:                                       *
*    PROCESSING FOR THE TABLE OR VIEW JUST READ               *
*                                                             *
***************************************************************
 PROCESS-INPUT.
*
     MOVE TNAME TO STMTTAB.
     MOVE STMTBLD TO STMTCHAR.
     MOVE +750 TO SQLN.
     EXEC SQL PREPARE SEL INTO :SQLDA FROM :STMTBUF  END-EXEC.
***************************************************************
*                                                             *
*    SET UP ADDRESSES IN THE SQLDA FOR DATA.                  *
*                                                             *
***************************************************************
     IF SQLD = ZERO THEN
        WRITE MSGREC FROM MSGNOCOL
                AFTER ADVANCING 2 LINES
        MOVE 'Y' TO ERR-FOUND
        GO TO IND-RESULT.
     MOVE ZERO TO ROWCOUNT.
     MOVE ZERO TO REC1-LEN.
     SET RECPTR TO IRECPTR.
     MOVE ONE TO I.
     PERFORM COLADDR UNTIL I > SQLD.
****************************************************************
*                                                              *
*    SET LENGTH OF OUTPUT RECORD.                              *
*    EXEC  SQL OPEN DT END-EXEC.                               *
*    DO WHILE SQLCODE IS 0.                                    *
*       EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.    *
*       ADD IN MARKERS TO DENOTE NULLS.                        *
*       WRITE THE DATA TO SYSREC01.                            *
*       INCREMENT DATA RECORD COUNTER.                         *
*    END.                                                      *
*                                                              *
****************************************************************
*                                         **OPEN CURSOR
     EXEC SQL OPEN DT  END-EXEC.
     PERFORM BLANK-REC.
     EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.
*                                          **NO ROWS FOUND
*                                          **PRINT ERROR MESSAGE
          IF SQLCODE = NOT-FOUND
             WRITE MSGREC FROM MSG-NOROW
                AFTER ADVANCING 2 LINES
```

```
              MOVE 'Y' TO ERR-FOUND
           ELSE
*                                          **WRITE ROW AND
*                                          **CONTINUE UNTIL
*                                          **NO MORE ROWS
              PERFORM WRITE-AND-FETCH
                 UNTIL SQLCODE IS NOT EQUAL TO ZERO.
*
     EXEC SQL WHENEVER NOT FOUND  GOTO CLOSEDT    END-EXEC.
*
 CLOSEDT.
     EXEC SQL CLOSE DT  END-EXEC.
*
*****************************************************************
*                                                               *
*    INDICATE THE RESULTS OF THE UNLOAD OPERATION.              *
*                                                               *
*****************************************************************
 IND-RESULT.
     IF ERR-FOUND = 'N' THEN
          MOVE TNAME TO TABLENAM
          MOVE ROWCOUNT TO ROWS
          WRITE MSGREC FROM UNLOADED
                AFTER ADVANCING 2 LINES
     ELSE
          WRITE MSGREC FROM MSG-OTHER-ERR
                AFTER ADVANCING 2 LINES
          MOVE +0012 TO RETURN-CODE
          GO TO PROG-END.
*
 WRITE-AND-FETCH.
*        ADD IN MARKERS TO DENOTE NULLS.
         MOVE ONE TO INDCOUNT.
         PERFORM NULLCHK UNTIL INDCOUNT = SQLD.
         MOVE REC1-LEN TO REC01-LEN.
         WRITE REC01 FROM LINKAREA-REC.
         ADD ONE TO ROWCOUNT.
         PERFORM BLANK-REC.
         EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.
*
 NULLCHK.
         IF IND(INDCOUNT) < 0 THEN
            SET ADDRESS OF LINKAREA-QMARK TO WORKINDPTR(INDCOUNT)
            MOVE QMARK TO INDREC.
         ADD ONE TO INDCOUNT.
*****************************************************
*    BLANK OUT RECORD TEXT FIRST                    *
*****************************************************
 BLANK-REC.
          MOVE ONE TO J.
          PERFORM BLANK-MORE UNTIL J > REC1-LEN.
 BLANK-MORE.
          MOVE ' ' TO REC1-CHAR(J).
          ADD  ONE TO J.
*
 COLADDR.
      SET SQLDATA(I) TO RECPTR.
*****************************************************************
*
*        DETERMINE THE LENGTH OF THIS COLUMN (COLUMN-LEN)
*        THIS DEPENDS UPON THE DATA TYPE.  MOST DATA TYPES HAVE
*        THE LENGTH SET, BUT VARCHAR, GRAPHIC, VARGRAPHIC, AND
*        DECIMAL DATA NEED TO HAVE THE BYTES CALCULATED.
*        THE NULL ATTRIBUTE MUST BE SEPARATED TO SIMPLIFY MATTERS.
*
*****************************************************************
      MOVE SQLLEN(I) TO COLUMN-LEN.
*        COLUMN-IND IS 0 FOR NO NULLS AND 1 FOR NULLS
      DIVIDE SQLTYPE(I) BY TWO GIVING DUMMY REMAINDER COLUMN-IND.
*        MYTYPE IS JUST THE SQLTYPE WITHOUT THE NULL BIT
      MOVE SQLTYPE(I) TO MYTYPE.
      SUBTRACT COLUMN-IND FROM MYTYPE.
*        SET THE COLUMN LENGTH, DEPENDENT UPON DATA TYPE
      EVALUATE MYTYPE
         WHEN     CHARTYPE  CONTINUE,
         WHEN     DATETYP   CONTINUE,
         WHEN     TIMETYP   CONTINUE,
         WHEN     TIMESTMP  CONTINUE,
         WHEN     FLOATYPE  CONTINUE,
         WHEN     VARCTYPE
               ADD TWO TO COLUMN-LEN,
         WHEN     VARLTYPE
```

```
                ADD TWO TO COLUMN-LEN,
          WHEN     GTYPE
                MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN,
          WHEN     VARGTYPE
                PERFORM CALC-VARG-LEN,
          WHEN     LVARGTYP
                PERFORM CALC-VARG-LEN,
          WHEN     HWTYPE
                MOVE TWO TO COLUMN-LEN,
          WHEN     INTTYPE
                MOVE FOUR TO COLUMN-LEN,
          WHEN     DECTYPE
                PERFORM CALC-DECIMAL-LEN,
          WHEN     OTHER
                PERFORM UNRECOGNIZED-ERROR,
       END-EVALUATE.
       ADD COLUMN-LEN TO RECNUM.
       ADD COLUMN-LEN TO REC1-LEN.
*****************************************************************
*                                                               *
*    IF THIS COLUMN CAN BE NULL, AN INDICATOR VARIABLE IS       *
*    NEEDED.  WE ALSO RESERVE SPACE IN THE OUTPUT RECORD TO     *
*    NOTE THAT THE VALUE IS NULL.                               *
*                                                               *
*****************************************************************
     MOVE ZERO TO IND(I).
     IF COLUMN-IND = ONE THEN
        SET SQLIND(I) TO ADDRESS OF IND(I)
        SET WORKINDPTR(I) TO RECPTR
        ADD ONE TO RECNUM
        ADD ONE TO REC1-LEN.
*
     ADD ONE TO I.
*        PERFORMED PARAGRAPH TO CALCULATE COLUMN LENGTH
*        FOR A DECIMAL DATA TYPE COLUMN
 CALC-DECIMAL-LEN.
         DIVIDE COLUMN-LEN BY 256 GIVING COLUMN-PREC
                  REMAINDER COLUMN-SCALE.
         MOVE COLUMN-PREC TO COLUMN-LEN.
         ADD ONE TO COLUMN-LEN.
         DIVIDE COLUMN-LEN BY TWO GIVING COLUMN-LEN.
*        PERFORMED PARAGRAPH TO CALCULATE COLUMN LENGTH
*        FOR A VARGRAPHIC DATA TYPE COLUMN
 CALC-VARG-LEN.
         MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN.
         ADD TWO TO COLUMN-LEN.
*        PERFORMED PARAGRAPH TO NOTE AN UNRECOGNIZED
*        DATA TYPE COLUMN
 UNRECOGNIZED-ERROR.
*
*        ERROR MESSAGE FOR UNRECOGNIZED DATA TYPE
*
         MOVE  SQLTYPE(I) TO TYPCOD
         MOVE 'Y' TO ERR-FOUND
         WRITE MSGREC FROM BADTYPE
                AFTER ADVANCING 2 LINES
         GO TO IND-RESULT.
*
*****************************************************
* SQL ERROR OCCURRED - GET MESSAGE                  *
*****************************************************
 DBERROR.
*                                         **SQL ERROR
          MOVE 'Y' TO ERR-FOUND.
          MOVE SQLCODE TO MSG-PRINT-CODE.
          IF SQLCODE < 0 THEN MOVE '-' TO MSG-MINUS.
          WRITE MSGREC FROM MSG-SQLERR
             AFTER ADVANCING 2 LINES.
          CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
          IF RETURN-CODE = ZERO
             PERFORM ERROR-PRINT VARYING ERROR-INDEX
                FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
          ELSE
*                                      **ERROR FOUND IN DSNTIAR
*                                      **PRINT ERROR MESSAGE
             MOVE RETURN-CODE TO RETCODE
             WRITE MSGREC FROM MSGRETCD
                AFTER ADVANCING 2 LINES.
          GO TO IND-RESULT.
*
*****************************************************
*    PRINT MESSAGE TEXT                             *
```

```
      *****************************************************
       ERROR-PRINT.
                WRITE MSGREC FROM ERROR-TEXT (ERROR-INDEX)
                   AFTER ADVANCING 1 LINE.
```

Related concepts

Program directories for Db2 11 (Db2 for z/OS in IBM Documentation)

Sample COBOL program with CONNECT statements

This example demonstrates how to access distributed data by using CONNECT statements in a COBOL program.

The following figure contains a sample COBOL program that uses two-phase commit to access distributed data.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TWOPHASE.
       AUTHOR.
       REMARKS.
      *****************************************************************
      *                                                               *
      * MODULE NAME = TWOPHASE                                        *
      *                                                               *
      * DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION USING               *
      *                    TWO PHASE COMMIT AND THE DRDA DISTRIBUTED  *
      *                    ACCESS METHOD WITH CONNECT STATEMENTS      *
      *                                                               *
      * COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1989        *
      * REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083         *
      *                                                               *
      * STATUS = VERSION 5                                            *
      *                                                               *
      * FUNCTION = THIS MODULE DEMONSTRATES DISTRIBUTED DATA ACCESS   *
      *            USING 2 PHASE COMMIT BY TRANSFERRING AN EMPLOYEE   *
      *            FROM ONE LOCATION TO ANOTHER.                      *
      *                                                               *
      *            NOTE: THIS PROGRAM ASSUMES THE EXISTENCE OF THE    *
      *                  TABLE SYSADM.EMP AT LOCATIONS STLEC1 AND     *
      *                  STLEC2.                                      *
      *                                                               *
      * MODULE TYPE = COBOL PROGRAM                                   *
      *    PROCESSOR   = DB2 PRECOMPILER, ENTERPRISE COBOL FOR Z/OS   *
      *    MODULE SIZE = SEE LINK EDIT                                *
      *    ATTRIBUTES  = NOT REENTRANT OR REUSABLE                    *
      *                                                               *
      * ENTRY POINT =                                                 *
      *    PURPOSE = TO ILLUSTRATE 2 PHASE COMMIT                     *
      *    LINKAGE = INVOKE FROM DSN RUN                              *
      *    INPUT   = NONE                                             *
      *    OUTPUT  =                                                  *
      *              SYMBOLIC LABEL/NAME = SYSPRINT                   *
      *              DESCRIPTION = PRINT OUT THE DESCRIPTION OF EACH  *
      *                STEP AND THE RESULTANT SQLCA                   *
      *                                                               *
      * EXIT NORMAL = RETURN CODE 0 FROM NORMAL COMPLETION            *
      *                                                               *
      * EXIT ERROR = NONE                                             *
      *                                                               *
      * EXTERNAL REFERENCES =                                         *
      *    ROUTINE SERVICES = NONE                                    *
      *    DATA-AREAS       = NONE                                    *
      *    CONTROL-BLOCKS   =                                         *
      *       SQLCA   -   SQL COMMUNICATION AREA                      *
      *                                                               *
      * TABLES = NONE                                                 *
      *                                                               *
      * CHANGE-ACTIVITY = NONE                                        *
      *                                                               *
      *                                                               *
      *                                                               *
```

```
      * PSEUDOCODE                                                    *
      *                                                               *
      *    MAINLINE.                                                  *
      *      Perform CONNECT-TO-SITE-1 to establish                   *
      *        a connection to the local connection.                  *
```

```
      *      If the previous operation was successful Then             *
      *        Do.                                                     *
      *        | Perform PROCESS-CURSOR-SITE-1 to obtain the           *
      *        |   information about an employee that is               *
      *        |   transferring to another location.                   *
      *        | If the information about the employee was obtained   *
      *        |   successfully Then                                   *
      *        |   Do.                                                 *
      *        |   | Perform UPDATE-ADDRESS to update the information *
      *        |   |   to contain current information about the       *
      *        |   |   employee.                                       *
      *        |   | Perform CONNECT-TO-SITE-2 to establish            *
      *        |   |   a connection to the site where the employee is *
      *        |   |   transferring to.                                *
      *        |   |   If the connection is established successfully  *
      *        |   |   Then                                            *
      *        |   |     Do.                                           *
      *        |   |     | Perform PROCESS-SITE-2 to insert the       *
      *        |   |     |   employee information at the location     *
      *        |   |     |   where the employee is transferring to.   *
      *        |   |     End if the connection was established        *
      *        |   |       successfully.                               *
      *        |   End if the employee information was obtained       *
      *        |     successfully.                                     *
      *        End if the previous operation was successful.          *
      *      Perform COMMIT-WORK to COMMIT the changes made to STLEC1 *
      *        and STLEC2.                                             *
      *                                                                *
      *    PROG-END.                                                   *
      *      Close the printer.                                        *
      *      Return.                                                   *
      *                                                                *
      *    CONNECT-TO-SITE-1.                                          *
      *      Provide a text description of the following step.        *
      *      Establish a connection to the location where the         *
      *        employee is transferring from.                          *
      *      Print the SQLCA out.                                      *
      *                                                                *
      *    PROCESS-CURSOR-SITE-1.                                      *
      *      Provide a text description of the following step.        *
      *      Open a cursor that will be used to retrieve information  *
      *        about the transferring employee from this site.        *
      *      Print the SQLCA out.                                      *
      *      If the cursor was opened successfully Then               *
      *        Do.                                                     *
      *        | Perform FETCH-DELETE-SITE-1 to retrieve and           *
      *        |   delete the information about the transferring      *
      *        |   employee from this site.                            *
      *        | Perform CLOSE-CURSOR-SITE-1 to close the cursor.     *
      *        End if the cursor was opened successfully.             *
      *                                                                *
```

```
      *    FETCH-DELETE-SITE-1.                                        *
      *      Provide a text description of the following step.        *
      *      Fetch information about the transferring employee.       *
      *      Print the SQLCA out.                                      *
      *      If the information was retrieved successfully Then       *
      *        Do.                                                     *
      *        | Perform DELETE-SITE-1 to delete the employee          *
      *        |   at this site.                                       *
      *        End if the information was retrieved successfully.     *
      *                                                                *
      *    DELETE-SITE-1.                                              *
      *      Provide a text description of the following step.        *
      *      Delete the information about the transferring employee   *
      *        from this site.                                         *
      *      Print the SQLCA out.                                      *
      *                                                                *
      *    CLOSE-CURSOR-SITE-1.                                        *
      *      Provide a text description of the following step.        *
      *      Close the cursor used to retrieve information about      *
      *        the transferring employee.                              *
      *      Print the SQLCA out.                                      *
      *                                                                *
      *    UPDATE-ADDRESS.                                             *
      *      Update the address of the employee.                       *
      *      Update the city of the employee.                          *
      *      Update the location of the employee.                      *
      *                                                                *
      *    CONNECT-TO-SITE-2.                                          *
      *      Provide a text description of the following step.        *
```

```
      *      Establish a connection to the location where the         *
      *        employee is transferring to.                           *
      *      Print the SQLCA out.                                     *
      *                                                               *
      *    PROCESS-SITE-2.                                            *
      *      Provide a text description of the following step.        *
      *      Insert the employee information at the location where    *
      *        the employee is being transferred to.                  *
      *      Print the SQLCA out.                                     *
      *                                                               *
      *    COMMIT-WORK.                                               *
      *      COMMIT all the changes made to STLEC1 and STLEC2.        *
      *                                                               *
      *****************************************************************


       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT PRINTER, ASSIGN TO S-OUT1.

       DATA DIVISION.
       FILE SECTION.
       FD  PRINTER
           RECORD CONTAINS 120 CHARACTERS
           DATA RECORD IS PRT-TC-RESULTS
           LABEL RECORD IS OMITTED.
       01  PRT-TC-RESULTS.
           03  PRT-BLANK              PIC X(120).
```

```
       WORKING-STORAGE SECTION.

      *****************************************************************
      * Variable declarations                                         *
      *****************************************************************

       01  H-EMPTBL.
           05  H-EMPNO   PIC X(6).
           05  H-NAME.
               49 H-NAME-LN   PIC S9(4) COMP-4.
               49 H-NAME-DA   PIC X(32).
           05  H-ADDRESS.
               49 H-ADDRESS-LN   PIC S9(4) COMP-4.
               49 H-ADDRESS-DA   PIC X(36).
           05  H-CITY.
               49 H-CITY-LN   PIC S9(4) COMP-4.
               49 H-CITY-DA   PIC X(36).
           05  H-EMPLOC   PIC X(4).
           05  H-SSNO     PIC X(11).
           05  H-BORN     PIC X(10).
           05  H-SEX      PIC X(1).
           05  H-HIRED    PIC X(10).
           05  H-DEPTNO   PIC X(3).
           05  H-JOBCODE  PIC S9(3)V COMP-3.
           05  H-SRATE    PIC S9(5) COMP.
           05  H-EDUC     PIC S9(5) COMP.
           05  H-SAL      PIC S9(6)V9(2) COMP-3.
           05  H-VALIDCHK PIC S9(6)V COMP-3.

       01  H-EMPTBL-IND-TABLE.
           02  H-EMPTBL-IND        PIC S9(4) COMP OCCURS 15 TIMES.

      *****************************************************************
      * Includes for the variables used in the COBOL standard         *
      * language procedures and the SQLCA.                            *
      *****************************************************************

           EXEC SQL INCLUDE COBSVAR END-EXEC.
           EXEC SQL INCLUDE SQLCA END-EXEC.

      *****************************************************************
      * Declaration for the table that contains employee information  *
      *****************************************************************

           EXEC SQL DECLARE SYSADM.EMP TABLE
               (EMPNO   CHAR(6) NOT NULL,
                NAME    VARCHAR(32),
                ADDRESS VARCHAR(36) ,
                CITY    VARCHAR(36) ,
                EMPLOC  CHAR(4) NOT NULL,
```

```
           SSNO    CHAR(11),
           BORN    DATE,
           SEX     CHAR(1),
           HIRED   CHAR(10),
           DEPTNO  CHAR(3) NOT NULL,
           JOBCODE DECIMAL(3),
           SRATE   SMALLINT,
           EDUC    SMALLINT,
```

```
           SAL     DECIMAL(8,2) NOT NULL,
           VALCHK  DECIMAL(6))
      END-EXEC.

*****************************************************************
* Constants                                                     *
*****************************************************************

 77  SITE-1                    PIC X(16) VALUE 'STLEC1'.
 77  SITE-2                    PIC X(16) VALUE 'STLEC2'.
 77  TEMP-EMPNO                PIC X(6)  VALUE '080000'.
 77  TEMP-ADDRESS-LN           PIC 99    VALUE 15.
 77  TEMP-CITY-LN              PIC 99    VALUE 18.

*****************************************************************
* Declaration of the cursor that will be used to retrieve       *
* information about a transferring employee                     *
*****************************************************************

      EXEC SQL DECLARE C1 CURSOR FOR
           SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
                  SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
                  SRATE, EDUC, SAL, VALCHK
           FROM   SYSADM.EMP
           WHERE EMPNO = :TEMP-EMPNO
      END-EXEC.

 PROCEDURE DIVISION.
 A101-HOUSE-KEEPING.
     OPEN OUTPUT PRINTER.

*****************************************************************
* An employee is transferring from location STLEC1 to STLEC2.   *
* Retrieve information about the employee from STLEC1, delete   *
* the employee from STLEC1 and insert the employee at STLEC2    *
* using the information obtained from STLEC1.                   *
*****************************************************************

 MAINLINE.
     PERFORM CONNECT-TO-SITE-1
     IF SQLCODE IS EQUAL TO 0
        PERFORM PROCESS-CURSOR-SITE-1
        IF SQLCODE IS EQUAL TO 0
           PERFORM UPDATE-ADDRESS
           PERFORM CONNECT-TO-SITE-2
           IF SQLCODE IS EQUAL TO 0
              PERFORM PROCESS-SITE-2.
     PERFORM COMMIT-WORK.
```

```
 PROG-END.
     CLOSE PRINTER.
     GOBACK.

*****************************************************************
* Establish a connection to STLEC1                              *
*****************************************************************

 CONNECT-TO-SITE-1.

     MOVE 'CONNECT TO STLEC1   ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
        CONNECT TO :SITE-1
     END-EXEC.
     PERFORM PTSQLCA.

*****************************************************************
* When a connection has been established successfully at STLEC1,*
* open the cursor that will be used to retrieve information     *
* about the transferring employee.                              *
*****************************************************************
```

```
 PROCESS-CURSOR-SITE-1.

     MOVE 'OPEN CURSOR C1      ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
        OPEN C1
     END-EXEC.
     PERFORM PTSQLCA.
     IF SQLCODE IS EQUAL TO ZERO
         PERFORM FETCH-DELETE-SITE-1
         PERFORM CLOSE-CURSOR-SITE-1.

*****************************************************************
* Retrieve information about the transferring employee.         *
* Provided that the employee exists, perform DELETE-SITE-1 to   *
* delete the employee from STLEC1.                              *
*****************************************************************

 FETCH-DELETE-SITE-1.

     MOVE 'FETCH C1      ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
        FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
     END-EXEC.
     PERFORM PTSQLCA.
     IF SQLCODE IS EQUAL TO ZERO
         PERFORM DELETE-SITE-1.
```

```
*****************************************************************
* Delete the employee from STLEC1.                              *
*****************************************************************

 DELETE-SITE-1.

     MOVE 'DELETE EMPLOYEE ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     MOVE 'DELETE EMPLOYEE     ' TO STNAME
     EXEC SQL
        DELETE FROM SYSADM.EMP
          WHERE EMPNO = :TEMP-EMPNO
     END-EXEC.
     PERFORM PTSQLCA.

*****************************************************************
* Close the cursor used to retrieve information about the       *
* transferring employee.                                        *
*****************************************************************

 CLOSE-CURSOR-SITE-1.

     MOVE 'CLOSE CURSOR C1     ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
        CLOSE C1
     END-EXEC.
     PERFORM PTSQLCA.

*****************************************************************
* Update certain employee information in order to make it       *
* current.                                                      *
*****************************************************************

 UPDATE-ADDRESS.
     MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
     MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
     MOVE TEMP-CITY-LN         TO H-CITY-LN.
     MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
     MOVE 'SJCA'               TO H-EMPLOC.

*****************************************************************
* Establish a connection to STLEC2                              *
*****************************************************************

 CONNECT-TO-SITE-2.

     MOVE 'CONNECT TO STLEC2   ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
```

```
           CONNECT TO :SITE-2
        END-EXEC.
        PERFORM PTSQLCA.
```

```
      *****************************************************************
      * Using the employee information that was retrieved from STLEC1 *
      * and updated previously, insert the employee at STLEC2.            *
      *****************************************************************

       PROCESS-SITE-2.

           MOVE 'INSERT EMPLOYEE     ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              INSERT INTO SYSADM.EMP VALUES
              (:H-EMPNO,
               :H-NAME,
               :H-ADDRESS,
               :H-CITY,
               :H-EMPLOC,
               :H-SSNO,
               :H-BORN,
               :H-SEX,
               :H-HIRED,
               :H-DEPTNO,
               :H-JOBCODE,
               :H-SRATE,
               :H-EDUC,
               :H-SAL,
               :H-VALIDCHK)
           END-EXEC.
           PERFORM PTSQLCA.

      *****************************************************************
      * COMMIT any changes that were made at STLEC1 and STLEC2.       *
      *****************************************************************

       COMMIT-WORK.

           MOVE 'COMMIT WORK          ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              COMMIT
           END-EXEC.
           PERFORM PTSQLCA.

      *****************************************************************
      * Include COBOL standard language procedures                    *
      *****************************************************************

       INCLUDE-SUBS.
           EXEC SQL INCLUDE COBSSUB END-EXEC.
```

Sample COBOL program using aliases for three-part names

You can access distributed data by using aliases for three-part names in a COBOL program.

The following sample program demonstrates distributed access data using aliases for three-part names with two-phase commit.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TWOPHASE.
       AUTHOR.
       REMARKS.
      *****************************************************************
      *                                                               *
      * MODULE NAME = TWOPHASE                                        *
      *                                                               *
      * DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION USING               *
      *                    TWO PHASE COMMIT AND DRDA WITH             *
      *                    ALIASES FOR THREE-PART NAMES               *
      *                                                               *
      * FUNCTION = THIS MODULE DEMONSTRATES DISTRIBUTED DATA ACCESS   *
      *            USING 2 PHASE COMMIT BY TRANSFERRING AN EMPLOYEE   *
      *            FROM ONE LOCATION TO ANOTHER.                      *
      *                                                               *
      *            NOTE: THIS PROGRAM ASSUMES THE EXISTENCE OF THE    *
```

```
      *                  TABLE SYSADM.ALLEMPLOYEES AT LOCATIONS STLEC1
      *                  AND STLEC2.                                    *
      *                                                                 *
      * MODULE TYPE = COBOL PROGRAM                                     *
      *    PROCESSOR   = DB2 PRECOMPILER, ENTERPRISE COBOL FOR Z/OS     *
      *    MODULE SIZE = SEE LINK EDIT                                  *
      *    ATTRIBUTES  = NOT REENTRANT OR REUSABLE                      *
      *                                                                 *
      * ENTRY POINT =                                                   *
      *    PURPOSE = TO ILLUSTRATE 2 PHASE COMMIT                       *
      *    LINKAGE = INVOKE FROM DSN RUN                                *
      *    INPUT   = NONE                                               *
      *    OUTPUT  =                                                    *
      *              SYMBOLIC LABEL/NAME = SYSPRINT                     *
      *              DESCRIPTION = PRINT OUT THE DESCRIPTION OF EACH    *
      *                STEP AND THE RESULTANT SQLCA                     *
      *                                                                 *
      * EXIT NORMAL = RETURN CODE 0 FROM NORMAL COMPLETION              *
      *                                                                 *
      * EXIT ERROR = NONE                                               *
      *                                                                 *
      * EXTERNAL REFERENCES =                                           *
      *    ROUTINE SERVICES = NONE                                      *
      *    DATA-AREAS       = NONE                                      *
      *    CONTROL-BLOCKS   =                                           *
      *       SQLCA   -   SQL COMMUNICATION AREA                        *
      *                                                                 *
      * TABLES = NONE                                                   *
      *                                                                 *
      * CHANGE-ACTIVITY = NONE                                          *
      *                                                                 *
      *                                                                 *
      *                                                                 *
      * PSEUDOCODE                                                      *
      *                                                                 *
      *    MAINLINE.                                                    *
      *      Perform PROCESS-CURSOR-SITE-1 to obtain the information    *
      *        about an employee that is transferring to another        *
      *        location.                                                *
      *      If the information about the employee was obtained         *
      *        successfully Then                                        *
      *        Do.                                                      *
      *        | Perform UPDATE-ADDRESS to update the information to    *
      *        |   contain current information about the employee.      *
      *        | Perform PROCESS-SITE-2 to insert the employee          *
      *        |   information at the location where the employee is    *
      *        |   transferring to.                                     *
      *        End if the employee information was obtained             *
      *          successfully.                                          *
      *      Perform COMMIT-WORK to COMMIT the changes made to STLEC1   *
      *        and STLEC2.                                              *
      *                                                                 *
      *    PROG-END.                                                    *
      *      Close the printer.                                         *
      *      Return.                                                    *
      *                                                                 *
      *    PROCESS-CURSOR-SITE-1.                                       *
      *      Provide a text description of the following step.         *
      *      Open a cursor that will be used to retrieve information    *
      *        about the transferring employee from this site.          *
      *      Print the SQLCA out.                                       *
      *      If the cursor was opened successfully Then                 *
      *        Do.                                                      *
      *        | Perform FETCH-DELETE-SITE-1 to retrieve and            *
      *        |   delete the information about the transferring        *
      *        |   employee from this site.                             *
      *        | Perform CLOSE-CURSOR-SITE-1 to close the cursor.       *
      *        End if the cursor was opened successfully.               *
      *                                                                 *
      *    FETCH-DELETE-SITE-1.                                         *
      *      Provide a text description of the following step.         *
      *      Fetch information about the transferring employee.         *
      *      Print the SQLCA out.                                       *
      *      If the information was retrieved successfully Then         *
      *        Do.                                                      *
      *        | Perform DELETE-SITE-1 to delete the employee           *
      *        |   at this site.                                        *
      *        End if the information was retrieved successfully.       *
      *                                                                 *
      *    DELETE-SITE-1.                                               *
      *      Provide a text description of the following step.         *
      *      Delete the information about the transferring employee     *
```

```
*        from this site.                                         *
*      Print the SQLCA out.                                      *
*                                                                *
*    CLOSE-CURSOR-SITE-1.                                        *
*      Provide a text description of the following step.        *
*      Close the cursor used to retrieve information about      *
*        the transferring employee.                              *
*      Print the SQLCA out.                                      *
*                                                                *
*    UPDATE-ADDRESS.                                             *
*      Update the address of the employee.                       *
*      Update the city of the employee.                          *
*      Update the location of the employee.                      *
*                                                                *
*    PROCESS-SITE-2.                                             *
*      Provide a text description of the following step.        *
*      Insert the employee information at the location where    *
*        the employee is being transferred to.                   *
*      Print the SQLCA out.                                      *
*                                                                *
*    COMMIT-WORK.                                                *
*      COMMIT all the changes made to STLEC1 and STLEC2.        *
*                                                                *
*****************************************************************


 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT PRINTER, ASSIGN TO S-OUT1.

 DATA DIVISION.
 FILE SECTION.
 FD  PRINTER
     RECORD CONTAINS 120 CHARACTERS
     DATA RECORD IS PRT-TC-RESULTS
     LABEL RECORD IS OMITTED.
 01  PRT-TC-RESULTS.
     03  PRT-BLANK              PIC X(120).

 WORKING-STORAGE SECTION.

*****************************************************************
* Variable declarations                                          *
*****************************************************************

 01  H-EMPTBL.
     05  H-EMPNO   PIC X(6).
     05  H-NAME.
         49 H-NAME-LN   PIC S9(4) COMP-4.
         49 H-NAME-DA   PIC X(32).
     05  H-ADDRESS.
         49 H-ADDRESS-LN   PIC S9(4) COMP-4.
         49 H-ADDRESS-DA   PIC X(36).
     05  H-CITY.
         49 H-CITY-LN   PIC S9(4) COMP-4.
         49 H-CITY-DA   PIC X(36).
     05  H-EMPLOC   PIC X(4).
     05  H-SSNO     PIC X(11).
     05  H-BORN     PIC X(10).
     05  H-SEX      PIC X(1).
     05  H-HIRED    PIC X(10).
     05  H-DEPTNO   PIC X(3).
     05  H-JOBCODE  PIC S9(3)V COMP-3.
     05  H-SRATE    PIC S9(5) COMP.
     05  H-EDUC     PIC S9(5) COMP.
     05  H-SAL      PIC S9(6)V9(2) COMP-3.
     05  H-VALIDCHK PIC S9(6)V COMP-3.
 01  H-EMPTBL-IND-TABLE.
     02  H-EMPTBL-IND        PIC S9(4) COMP OCCURS 15 TIMES.

*****************************************************************
* Includes for the variables used in the COBOL standard         *
* language procedures and the SQLCA.                             *
*****************************************************************

     EXEC SQL INCLUDE COBSVAR END-EXEC.
     EXEC SQL INCLUDE SQLCA END-EXEC.

*****************************************************************
* Declaration for the table that contains employee information  *
*****************************************************************
```

```
     EXEC SQL DECLARE SYSADM.ALLEMPLOYEES TABLE
         (EMPNO   CHAR(6) NOT NULL,
          NAME    VARCHAR(32),
          ADDRESS VARCHAR(36) ,
          CITY    VARCHAR(36) ,
          EMPLOC  CHAR(4) NOT NULL,
          SSNO    CHAR(11),
          BORN    DATE,
          SEX     CHAR(1),
          HIRED   CHAR(10),
          DEPTNO  CHAR(3) NOT NULL,
          JOBCODE DECIMAL(3),
          SRATE   SMALLINT,
          EDUC    SMALLINT,
          SAL     DECIMAL(8,2) NOT NULL,
          VALCHK  DECIMAL(6))
     END-EXEC.

*****************************************************************
* Constants                                                     *
*****************************************************************

 77  TEMP-EMPNO                PIC X(6)  VALUE '080000'.
 77  TEMP-ADDRESS-LN           PIC 99    VALUE 15.
 77  TEMP-CITY-LN              PIC 99    VALUE 18.

*****************************************************************
* Declaration of the cursor that will be used to retrieve       *
* information about a transferring employee                     *
* EC1EMP is the alias for STLEC1.SYSADM.ALLEMPLOYEES            *
*****************************************************************

     EXEC SQL DECLARE C1 CURSOR FOR
          SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
                 SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
                 SRATE, EDUC, SAL, VALCHK
          FROM   EC1EMP
          WHERE EMPNO = :TEMP-EMPNO
     END-EXEC.
 PROCEDURE DIVISION.
 A101-HOUSE-KEEPING.
     OPEN OUTPUT PRINTER.

*****************************************************************
* An employee is transferring from location STLEC1 to STLEC2.   *
* Retrieve information about the employee from STLEC1, delete   *
* the employee from STLEC1 and insert the employee at STLEC2    *
* using the information obtained from STLEC1.                   *
*****************************************************************

 MAINLINE.
     PERFORM PROCESS-CURSOR-SITE-1
     IF SQLCODE IS EQUAL TO 0
        PERFORM UPDATE-ADDRESS
        PERFORM PROCESS-SITE-2.
     PERFORM COMMIT-WORK.

 PROG-END.
     CLOSE PRINTER.
     GOBACK.

*****************************************************************
* Open the cursor that will be used to retrieve information     *
* about the transferring employee.                              *
*****************************************************************

 PROCESS-CURSOR-SITE-1.

     MOVE 'OPEN CURSOR C1      ' TO STNAME
     WRITE PRT-TC-RESULTS FROM STNAME
     EXEC SQL
        OPEN C1
     END-EXEC.
     PERFORM PTSQLCA.
     IF SQLCODE IS EQUAL TO ZERO
         PERFORM FETCH-DELETE-SITE-1
         PERFORM CLOSE-CURSOR-SITE-1.

*****************************************************************
* Retrieve information about the transferring employee.         *
* Provided that the employee exists, perform DELETE-SITE-1 to   *
```

```
      * delete the employee from STLEC1.                              *
      *****************************************************************

       FETCH-DELETE-SITE-1.

           MOVE 'FETCH C1       ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
           END-EXEC.           PERFORM PTSQLCA.
           IF SQLCODE IS EQUAL TO ZERO
               PERFORM DELETE-SITE-1.

      *****************************************************************
      * Delete the employee from STLEC1.                              *
      *****************************************************************

       DELETE-SITE-1.

           MOVE 'DELETE EMPLOYEE ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           MOVE 'DELETE EMPLOYEE     ' TO STNAME
           EXEC SQL
              DELETE FROM EC1EMP
                WHERE EMPNO = :TEMP-EMPNO
           END-EXEC.
           PERFORM PTSQLCA.

      *****************************************************************
      * Close the cursor used to retrieve information about the       *
      * transferring employee.                                        *
      *****************************************************************

       CLOSE-CURSOR-SITE-1.

           MOVE 'CLOSE CURSOR C1     ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              CLOSE C1
           END-EXEC.
           PERFORM PTSQLCA.

      *****************************************************************
      * Update certain employee information in order to make it       *
      * current.                                                      *
      *****************************************************************

       UPDATE-ADDRESS.
           MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
           MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
           MOVE TEMP-CITY-LN         TO H-CITY-LN.
           MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
           MOVE 'SJCA'               TO H-EMPLOC.
       ****************************************************************
      * Using the employee information that was retrieved from STLEC1 *
      * and updated previously, insert the employee at STLEC2.        *
      * EC2EMP is the alias for STLEC2.SYSADM.ALLEMPLOYEES            *
      *****************************************************************

       PROCESS-SITE-2.

           MOVE 'INSERT EMPLOYEE     ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              INSERT INTO EC2EMP VALUES
              (:H-EMPNO,
               :H-NAME,
               :H-ADDRESS,
               :H-CITY,
               :H-EMPLOC,
               :H-SSNO,
               :H-BORN,
               :H-SEX,
               :H-HIRED,
               :H-DEPTNO,
               :H-JOBCODE,
               :H-SRATE,
               :H-EDUC,
               :H-SAL,
               :H-VALIDCHK)
           END-EXEC.
           PERFORM PTSQLCA.
```

```
      *****************************************************************
      * COMMIT any changes that were made at STLEC1 and STLEC2.       *
      *****************************************************************

       COMMIT-WORK.

           MOVE 'COMMIT WORK         ' TO STNAME
           WRITE PRT-TC-RESULTS FROM STNAME
           EXEC SQL
              COMMIT
           END-EXEC.
           PERFORM PTSQLCA.

      *****************************************************************
      * Include COBOL standard language procedures                    *
      *****************************************************************

       INCLUDE-SUBS.
           EXEC SQL INCLUDE COBSSUB END-EXEC.
```

Example COBOL stored procedure with a GENERAL WITH NULLS linkage convention

You can call a stored procedure that uses the GENERAL WITH NULLS linkage convention from a COBOL program.

This example stored procedure does the following:

- Searches the Db2 SYSIBM.SYSROUTINES catalog table for a row that matches the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.
- Searches the Db2 catalog table SYSTABLES for all tables in which the value of CREATOR matches the value of input parameter SCHEMA. The stored procedure uses a cursor to return the table names.

The linkage convention for this stored procedure is GENERAL WITH NULLS.

The output parameters from this stored procedure contain the SQLCODE from the SELECT operation, and the value of the RUNOPTS column retrieved from the SYSIBM.SYSROUTINES table.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE COBOL
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
  RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 2
  COMMIT ON RETURN NO;
```

The following example is a COBOL stored procedure with linkage convention GENERAL WITH NULLS.

```
 CBL RENT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. GETPRML.
 AUTHOR. EXAMPLE.
 DATE-WRITTEN.   03/25/98.

 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.
*
 WORKING-STORAGE SECTION.
*
```

```
     EXEC SQL INCLUDE SQLCA END-EXEC.
*
***************************************************
*   DECLARE A HOST VARIABLE TO HOLD INPUT SCHEMA
***************************************************
 01  INSCHEMA PIC X(8).
***************************************************
*   DECLARE CURSOR FOR RETURNING RESULT SETS
***************************************************
*
     EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
       SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
     END-EXEC.
*
 LINKAGE SECTION.
***************************************************
*   DECLARE THE INPUT PARAMETERS FOR THE PROCEDURE
***************************************************
 01  PROCNM  PIC X(18).
 01  SCHEMA  PIC X(8).
***************************************************
*   DECLARE THE OUTPUT PARAMETERS FOR THE PROCEDURE
***************************************************
 01  OUT-CODE PIC S9(9) USAGE BINARY.
 01  PARMLST.
     49 PARMLST-LEN  PIC S9(4) USAGE BINARY.
     49 PARMLST-TEXT PIC X(254).
***************************************************
*   DECLARE THE STRUCTURE CONTAINING THE NULL
*   INDICATORS FOR THE INPUT AND OUTPUT PARAMETERS.
***************************************************
 01  IND-PARM.
     03 PROCNM-IND   PIC S9(4) USAGE BINARY.
     03 SCHEMA-IND   PIC S9(4) USAGE BINARY.
     03 OUT-CODE-IND PIC S9(4) USAGE BINARY.
     03 PARMLST-IND  PIC S9(4) USAGE BINARY.
```

```
 PROCEDURE DIVISION USING PROCNM, SCHEMA,
           OUT-CODE, PARMLST, IND-PARM.
*******************************************************
* If any input parameter is null, return a null value
* for PARMLST and set the output return code to 9999.
*******************************************************
      IF PROCNM-IND < 0 OR
         SCHEMA-IND < 0
           MOVE 9999 TO OUT-CODE
           MOVE 0 TO OUT-CODE-IND
           MOVE -1 TO PARMLST-IND
      ELSE
*******************************************************
* Issue the SQL SELECT against the SYSIBM.SYSROUTINES
* DB2 catalog table.
*******************************************************
     EXEC SQL
       SELECT RUNOPTS INTO :PARMLST
       FROM SYSIBM.SYSROUTINES
       WHERE NAME=:PROCNM AND
       SCHEMA=:SCHEMA
     END-EXEC
           MOVE 0 TO PARMLST-IND
*******************************************************
*  COPY SQLCODE INTO THE OUTPUT PARAMETER AREA
*******************************************************
           MOVE SQLCODE TO OUT-CODE
           MOVE 0 TO OUT-CODE-IND.
*
*******************************************************
* OPEN CURSOR C1 TO CAUSE DB2 TO RETURN A RESULT SET
* TO THE CALLER.
*******************************************************
     EXEC SQL OPEN C1
     END-EXEC.
 PROG-END.
     GOBACK.
```

Example COBOL stored procedure with a GENERAL linkage convention

You can call a stored procedure that uses the GENERAL linkage convention from a COBOL program.

This example stored procedure does the following:

- Searches the catalog table SYSROUTINES for a row matching the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.
- Searches the Db2 catalog table SYSTABLES for all tables in which the value of CREATOR matches the value of input parameter SCHEMA. The stored procedure uses a cursor to return the table names.

This stored procedure is able to return a NULL value for the output host variables.

The linkage convention for this stored procedure is GENERAL.

The output parameters from this stored procedure contain the SQLCODE from the SELECT operation, and the value of the RUNOPTS column retrieved from the SYSROUTINES table.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE COBOL
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL
  STAY RESIDENT NO
  RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 2
  COMMIT ON RETURN NO;
```

```
 CBL RENT
 IDENTIFICATION DIVISION.
 PROGRAM-ID. GETPRML.
 AUTHOR. EXAMPLE.
 DATE-WRITTEN.   03/25/98.

 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.

 WORKING-STORAGE SECTION.

     EXEC SQL INCLUDE SQLCA END-EXEC.
***************************************************
*   DECLARE A HOST VARIABLE TO HOLD INPUT SCHEMA
***************************************************
 01  INSCHEMA PIC X(8).

***************************************************
*   DECLARE CURSOR FOR RETURNING RESULT SETS
***************************************************
*
     EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
       SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
     END-EXEC.
*
 LINKAGE SECTION.
***************************************************
*   DECLARE THE INPUT PARAMETERS FOR THE PROCEDURE
***************************************************
 01  PROCNM  PIC X(18).
 01  SCHEMA  PIC X(8).
*******************************************************
*   DECLARE THE OUTPUT PARAMETERS FOR THE PROCEDURE
*******************************************************
 01  OUT-CODE PIC S9(9) USAGE BINARY.
 01  PARMLST.
```

```
     49 PARMLST-LEN  PIC S9(4) USAGE BINARY.
     49 PARMLST-TEXT PIC X(254).

 PROCEDURE DIVISION USING PROCNM, SCHEMA,
           OUT-CODE, PARMLST.
```

```
*******************************************************
* Issue the SQL SELECT against the SYSIBM.SYSROUTINES
* DB2 catalog table.
*******************************************************
     EXEC SQL
       SELECT RUNOPTS INTO :PARMLST
         FROM SYSIBM.ROUTINES
         WHERE NAME=:PROCNM AND
         SCHEMA=:SCHEMA
     END-EXEC.

*******************************************************
*  COPY SQLCODE INTO THE OUTPUT PARAMETER AREA
*******************************************************
      MOVE SQLCODE TO OUT-CODE.
*******************************************************
* OPEN CURSOR C1 TO CAUSE DB2 TO RETURN A RESULT SET
* TO THE CALLER.
*******************************************************
     EXEC SQL OPEN C1
     END-EXEC.
 PROG-END.
      GOBACK.
```

Example COBOL program that calls a stored procedure

You can call the GETPRML stored procedure that uses the GENERAL WITH NULLS linkage convention from a COBOL program on a z/OS system.

Because the stored procedure returns result sets, this program checks for result sets and retrieves the contents of the result sets. The following figure contains the example COBOL program that calls the GETPRML stored procedure.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    CALPRML.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT REPOUT
            ASSIGN TO UT-S-SYSPRINT.

 DATA DIVISION.
 FILE SECTION.
 FD  REPOUT
         RECORD CONTAINS 127 CHARACTERS
         LABEL RECORDS ARE OMITTED
         DATA RECORD IS REPREC.
 01  REPREC                     PIC X(127).

 WORKING-STORAGE SECTION.
*****************************************************
* MESSAGES FOR SQL CALL                             *
*****************************************************
 01  SQLREC.
         02  BADMSG    PIC X(34) VALUE
               ' SQL CALL FAILED DUE TO SQLCODE = '.
         02  BADCODE   PIC +9(5) USAGE DISPLAY.
         02  FILLER    PIC X(80) VALUE SPACES.
 01  ERRMREC.
         02  ERRMMSG   PIC X(12) VALUE ' SQLERRMC = '.
         02  ERRMCODE  PIC X(70).
         02  FILLER    PIC X(38) VALUE SPACES.
 01  CALLREC.
         02  CALLMSG   PIC X(28) VALUE
               ' GETPRML FAILED DUE TO RC = '.
         02  CALLCODE  PIC +9(5) USAGE DISPLAY.
         02  FILLER    PIC X(42) VALUE SPACES.
 01  RSLTREC.
         02  RSLTMSG   PIC X(15) VALUE
```

```
                ' TABLE NAME IS '.
          02  TBLNAME   PIC X(18) VALUE SPACES.
          02  FILLER    PIC X(87) VALUE SPACES.
```

```
*****************************************************
* WORK AREAS                                        *
*****************************************************
 01  PROCNM                      PIC X(18).
 01  SCHEMA                      PIC X(8).
 01  OUT-CODE                    PIC S9(9) USAGE COMP.
 01  PARMLST.
     49 PARMLEN          PIC S9(4) USAGE COMP.
     49 PARMTXT          PIC X(254).
 01  PARMBUF REDEFINES PARMLST.
     49 PARBLEN          PIC S9(4) USAGE COMP.
     49 PARMARRY         PIC X(127) OCCURS 2 TIMES.
 01  NAME.
     49 NAMELEN          PIC S9(4) USAGE COMP.
     49 NAMETXT          PIC X(18).
 77  PARMIND             PIC S9(4) COMP.
 77  I                   PIC S9(4) COMP.
 77  NUMLINES            PIC S9(4) COMP.
*****************************************************
* DECLARE A RESULT SET LOCATOR FOR THE RESULT SET   *
* THAT IS RETURNED.                                 *
*****************************************************
 01  LOC                 USAGE SQL TYPE IS
                         RESULT-SET-LOCATOR VARYING.

*****************************************************
* SQL INCLUDE FOR SQLCA                             *
*****************************************************
     EXEC SQL INCLUDE SQLCA  END-EXEC.

 PROCEDURE DIVISION.
*------------------
 PROG-START.
          OPEN OUTPUT REPOUT.
*                   OPEN OUTPUT FILE
          MOVE 'DSN8EP2           ' TO PROCNM.
*                   INPUT PARAMETER -- PROCEDURE TO BE FOUND
          MOVE SPACES TO SCHEMA.
*                   INPUT PARAMETER -- SCHEMA IN SYSROUTINES
          MOVE -1 TO PARMIND.
*                   THE PARMLST PARAMETER IS AN OUTPUT PARM.
*                   MARK PARMLST PARAMETER AS NULL, SO THE DB2
*                   REQUESTER DOES NOT HAVE TO SEND THE ENTIRE
*                   PARMLST VARIABLE TO THE SERVER.  THIS
*                   HELPS REDUCE NETWORK I/O TIME, BECAUSE
*                   PARMLST IS FAIRLY LARGE.
      EXEC SQL
         CALL GETPRML(:PROCNM,
                    :SCHEMA,
                    :OUT-CODE,
                    :PARMLST INDICATOR :PARMIND)
      END-EXEC.
```

```
*                   MAKE THE CALL
          IF SQLCODE NOT EQUAL TO +466 THEN
*                   IF CALL RETURNED BAD SQLCODE
            MOVE SQLCODE TO BADCODE
            WRITE REPREC FROM SQLREC
            MOVE SQLERRMC TO ERRMCODE
            WRITE REPREC FROM ERRMREC
          ELSE
            PERFORM GET-PARMS
            PERFORM GET-RESULT-SET.
 PROG-END.
          CLOSE REPOUT.
*                   CLOSE OUTPUT FILE
          GOBACK.
 PARMPRT.
          MOVE SPACES TO REPREC.
          WRITE REPREC FROM PARMARRY(I)
             AFTER ADVANCING 1 LINE.
 GET-PARMS.
*                   IF THE CALL WORKED,
      IF OUT-CODE NOT EQUAL TO 0 THEN
*                   DID GETPRML HIT AN ERROR?
        MOVE OUT-CODE TO CALLCODE
```

```
        WRITE REPREC FROM CALLREC
      ELSE
*                   EVERYTHING WORKED
         DIVIDE 127 INTO PARMLEN GIVING NUMLINES ROUNDED
*                   FIND OUT HOW MANY LINES TO PRINT
         PERFORM PARMPRT VARYING I
           FROM 1 BY 1 UNTIL I GREATER THAN NUMLINES.
 GET-RESULT-SET.
*****************************************************
* ASSUME YOU KNOW THAT ONE RESULT SET IS RETURNED,  *
* AND YOU KNOW THE FORMAT OF THAT RESULT SET.       *
* ALLOCATE A CURSOR FOR THE RESULT SET, AND FETCH   *
* THE CONTENTS OF THE RESULT SET.                   *
*****************************************************
      EXEC SQL ASSOCIATE LOCATORS (:LOC)
        WITH PROCEDURE GETPRML
      END-EXEC.
*                   LINK THE RESULT SET TO THE LOCATOR
     EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC
     END-EXEC.
*                   LINK THE CURSOR TO THE RESULT SET
     PERFORM GET-ROWS VARYING I
      FROM 1 BY 1 UNTIL SQLCODE EQUAL TO +100.
 GET-ROWS.
     EXEC SQL FETCH C1 INTO :NAME
     END-EXEC.
    MOVE NAME TO TBLNAME.
    WRITE REPREC FROM RSLTREC
      AFTER ADVANCING 1 LINE.
```

Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL

COBOL programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

About this task

If you specify the SQL processing option STDSQL(YES), do not define an SQLCA. If you do, Db2 ignores your SQLCA, and your SQLCA definition causes compile-time errors. If you specify the SQL processing option STDSQL(NO), include an SQLCA explicitly.

For COBOL programs, when you specify STDSQL(YES), you must declare an SQLCODE variable. Db2 declares an SQLCA area for you in the WORKING-STORAGE SECTION. Db2 controls the structure and location of the SQLCA.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Procedure

Choose one of the following actions:

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>To define the SQL communications area:</td><td>a. Code the SQLCA directly in the program or use the following SQL INCLUDE statement to request a standard SQLCA declaration:

<code>EXEC SQL INCLUDE SQLCA</code>

You can specify INCLUDE SQLCA or a declaration for SQLCODE wherever you can specify a 77 level or a record description entry in the WORKING-STORAGE SECTION.

Db2 sets the SQLCODE and SQLSTATE values in the SQLCA after each SQL statement executes. Your application should check these values to determine whether the last SQL statement was successful.</td></tr>
</table>

Option	Description
To declare SQLCODE and SQLSTATE host variables:	a. Declare the SQLCODE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as PIC S9(9) BINARY, PIC S9(9) COMP-4, PIC S9(9) COMP-5, or PICTURE S9(9) COMP. When you use the Db2 precompiler, you can declare a stand-alone SQLCODE variable in either the WORKING-STORAGE SECTION or LINKAGE SECTION. When you use the Db2 coprocessor, you can declare a stand-alone SQLCODE variable in the WORKING-STORAGE SECTION, LINKAGE SECTION or LOCAL-STORAGE SECTION. b. Declare the SQLSTATE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as PICTURE X(5). **Restriction:** Do not declare an SQLSTATE variable as an element of a structure. **Requirement:** After you declare the SQLCODE and SQLSTATE variables, ensure that all SQL statements in the program are within the scope of the declaration of these variables.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas (SQLDA) in COBOL

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Perform one of the following actions:

- Code the SQLDA declarations directly in your program. When you use the Db2 precompiler, you must place SQLDA declarations in the WORKING-STORAGE SECTION or LINKAGE SECTION of your program, wherever you can specify a record description entry in that section. When you use the Db2 coprocessor, you must place SQLDA declarations in the WORKING-STORAGE SECTION, LINKAGE SECTION or LOCAL-STORAGE SECTION of your program, wherever you can specify a record description entry in that section.
- Call a subroutine that is written in C, PL/I, or assembler language and that uses the INCLUDE SQLDA statement to define the SQLDA. The subroutine can also include SQL statements for any dynamic SQL functions that you need.

Restriction:

- You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Declaring host variables and indicator variables in COBOL

You can use host variables, host-variable arrays, and host structures in SQL statements in your program to pass data between Db2 and your application.

Procedure

To declare host variables, host-variable arrays, and host structures:

1. Declare the variables according to the following rules and guidelines:
   - You must explicitly declare all host variables and host-variable arrays that are used in SQL statements in the WORKING-STORAGE SECTION or LINKAGE SECTION of your program's DATA DIVISION.
   - You must explicitly declare each host variable and host-variable array before using them in an SQL statement.
   - You can specify OCCURS when defining an indicator structure, a host-variable array, or an indicator variable array. You cannot specify OCCURS for any other type of host variable.
   - You cannot implicitly declare any host variables through default typing or by using the IMPLICIT statement.
   - If you specify the ONEPASS SQL processing option, you must explicitly declare each host variable and each host-variable array before using them in an SQL statement. If you specify the TWOPASS precompiler option, you must declare each host variable before using it in the DECLARE CURSOR statement.
   - If you specify the STDSQL(YES) SQL processing option, you must precede the host language statements that define the host variables and host-variable arrays with the BEGIN DECLARE SECTION statement and follow the host language statements with the END DECLARE SECTION statement. Otherwise, these statements are optional.
   - Ensure that any SQL statement that uses a host variable or host-variable array is within the scope of the statement that declares that variable or array.
   - If you are using the Db2 precompiler, ensure that the names of host variables and host-variable arrays are unique within the program, even if the variables and variable arrays are in different blocks, classes, procedures, functions, or subroutines. You can qualify the names with a structure name to make them unique.
2. Optional: Define any associated indicator variables, arrays, and structures.

Related tasks

Declaring host variables and indicator variables

You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Host variables in COBOL

In COBOL programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set and table locators and LOB and XML file reference variables.

Restrictions:

- Only some of the valid COBOL declarations are valid host variable declarations. If the declaration for a variable is not valid, any SQL statement that references the variable might result in the message UNDECLARED HOST VARIABLE.
- You can not use locators as column types.

  The following locator data types are COBOL data types and SQL data types:

  – Result set locator
  – Table locator
  – LOB locators
  – LOB file reference variables
- One or more REDEFINES entries can follow any level 77 data description entry. However, you cannot use the names in these entries in SQL statements. Entries with the name FILLER are ignored.

Recommendations:

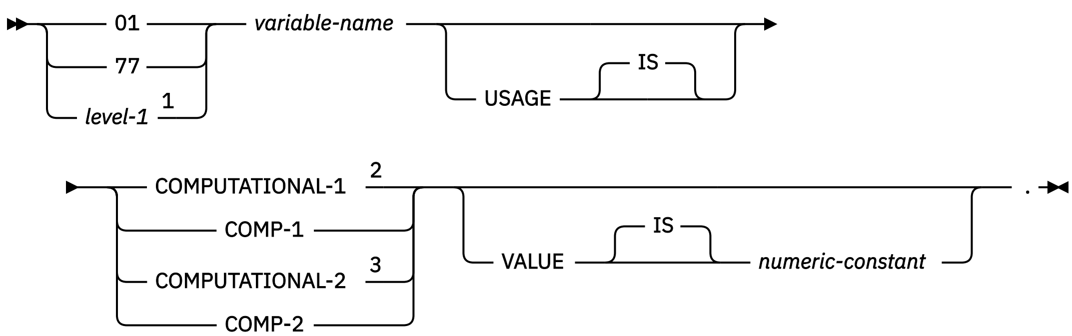
- Be careful of overflow. For example, suppose that you retrieve an INTEGER column value into a PICTURE S9(4) host variable and the column value is larger than 32767 or smaller than -32768. You get an overflow warning or an error, depending on whether you specify an indicator variable.
- Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a PICTURE X(70) host variable, the rightmost 10 characters of the retrieved string are truncated. Retrieving a double precision floating-point or decimal column value into a PIC S9(8) COMP host variable removes any fractional part of the value. Similarly, retrieving a column value with DECIMAL data type into a COBOL decimal variable with a lower precision might truncate the value.
- If your varying-length string host variables receive values whose length is greater than 9999 bytes, compile the applications in which you use those host variables with the option TRUNC(BIN). TRUNC(BIN) lets the length field for the string receive a value of up to 32767 bytes.

Numeric host variables

You can specify the following forms of numeric host variables:

- Floating-point numbers
- Integers and small integers
- Decimal numbers

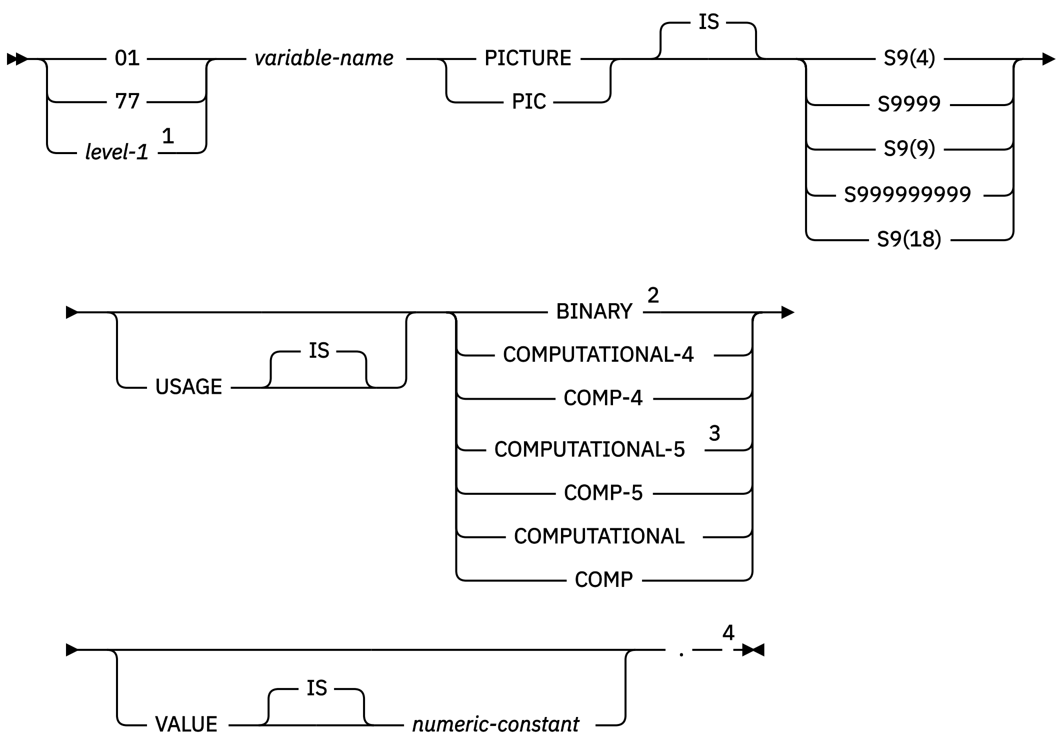
The following diagram shows the syntax for declaring floating-point or real host variables.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] COMPUTATIONAL-1 and COMP-1 are equivalent.
[3] COMPUTATIONAL-2 and COMP-2 are equivalent.

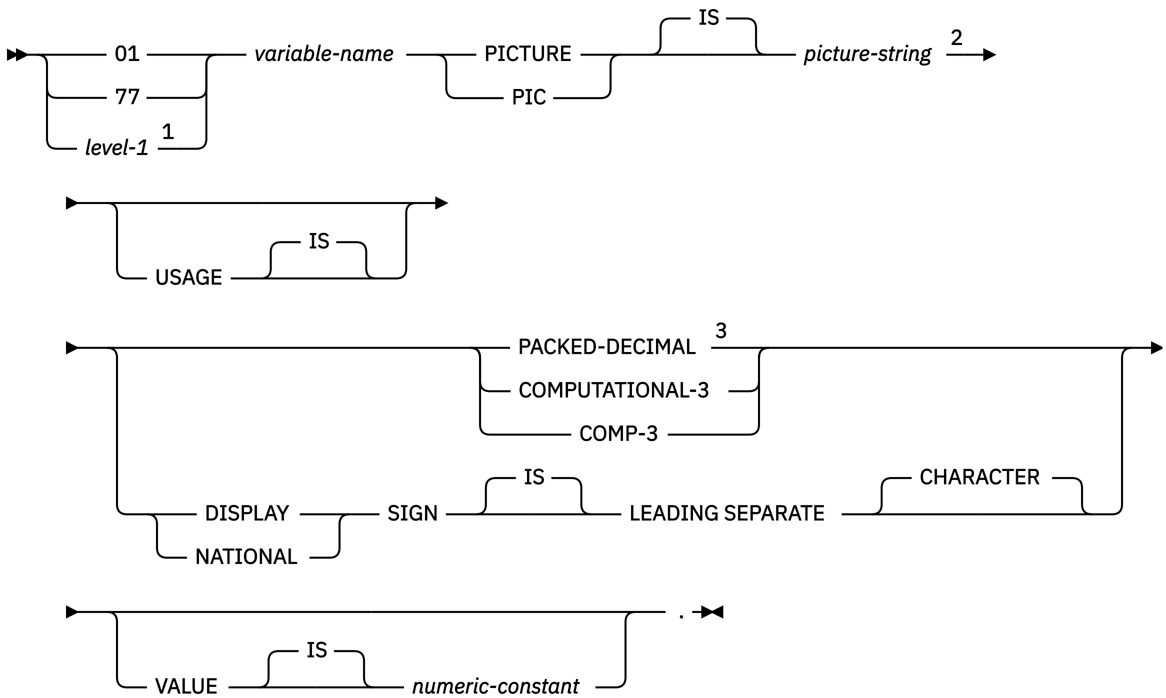
The following diagram shows the syntax for declaring integer and small integer host variables.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] The COBOL binary integer data types BINARY, COMPUTATIONAL, COMP, COMPUTATIONAL-4, and COMP-4 are equivalent.
[3] COMPUTATIONAL-5 (and COMP-5) are equivalent to the other COBOL binary integer data types if you compile the other data types with TRUNC(BIN).
[4] Any specification for scale is ignored.

The following diagram shows the syntax for declaring decimal host variables.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

[2] The *picture-string* that is associated with SIGN LEADING SEPARATE must have the form S9(*i*)V9(*d*) (or S9...9V9...9, with *i* and *d* instances of 9 or S9...9V with *i* instances of 9).

[3] PACKED-DECIMAL, COMPUTATIONAL-3, and COMP-3 are equivalent. The *picture-string* that is that is associated with these types must have the form S9(*i*)V9(*d*) (or S9...9V9...9, with *i* and *d* instances of 9) or S9(*i*)V.

In COBOL, you declare the SMALLINT and INTEGER data types as a number of decimal digits. Db2 uses the full size of the integers (in a way that is similar to processing with the TRUNC(BIN) compiler option) and can place larger values in the host variable than would be allowed in the specified number of digits in the COBOL declaration. If you compile with TRUNC(OPT) or TRUNC(STD), ensure that the size of numbers in your application is within the declared number of digits.

For small integers that can exceed 9999, use S9(4) COMP-5 or compile with TRUNC(BIN). For large integers that can exceed 999,999,999, use S9(10) COMP-3 to obtain the decimal data type. If you use COBOL for integers that exceed the COBOL PICTURE, specify the column as decimal to ensure that the data types match and perform well.

If you are using a COBOL compiler that does not support decimal numbers of more than 18 digits, use one of the following data types to hold values of greater than 18 digits:

- A decimal variable with a precision less than or equal to 18, if the actual data values fit. If you retrieve a decimal value into a decimal variable with a scale that is less than the source column in the database, the fractional part of the value might be truncated.
- An integer or a floating-point variable, which converts the value. If you use an integer variable, you lose the fractional part of the number. If the decimal number might exceed the maximum value for an integer or if you want to preserve a fractional value, use a floating-point variable. Floating-point numbers are approximations of real numbers. Therefore, when you assign a decimal number to a floating-point variable, the result might be different from the original number.
- A character-string host variable. Use the CHAR function to retrieve a decimal value into it.

Restriction: The SQL data type DECFLOAT has no equivalent in COBOL.

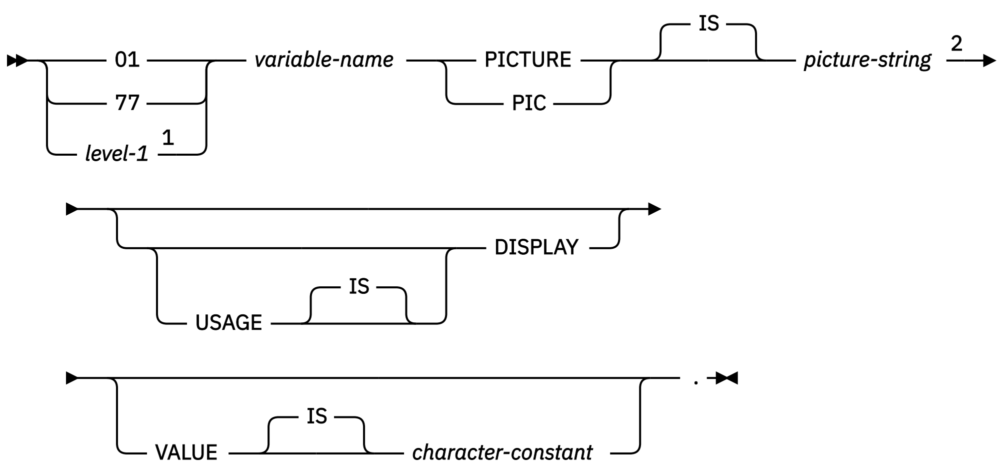
Character host variables

You can specify the following forms of character host variables:

- Fixed-length strings
- Varying-length strings
- CLOBs

The following diagrams show the syntax for forms other than CLOBs.

The following diagram shows the syntax for declaring fixed-length character host variables.

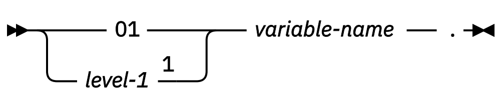


Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

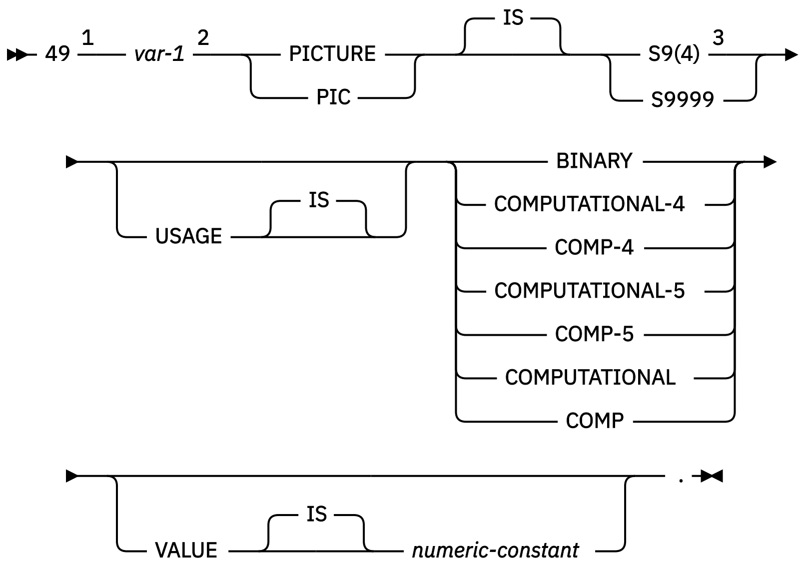
[2] The *picture-string* that is associated with these forms must be X(*m*) (or XX...X, with *m* instances of X), where *m* is up to COBOL's limitation. However, the maximum length of the CHAR data type (fixed-length character string) in Db2 is 255 bytes.

The following diagrams show the syntax for declaring varying-length character host variables.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

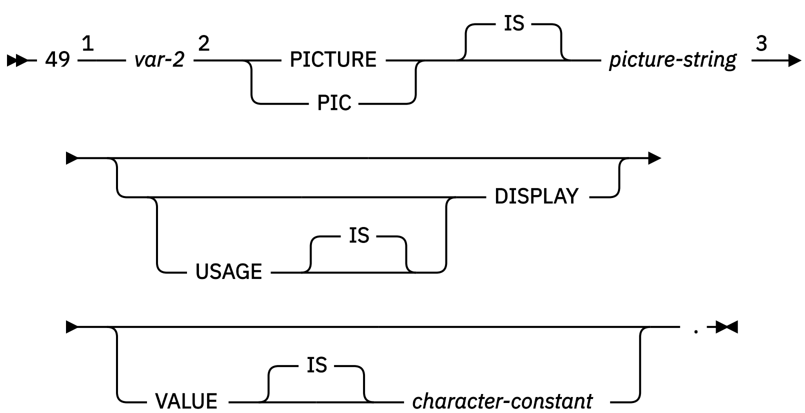


Notes:

[1] You cannot use an intervening REDEFINE at level 49.

[2] You cannot directly reference *var-1* as a host variable.

[3] Db2 uses the full length of the S9(4) BINARY variable even though COBOL with TRUNC(STD) recognizes values up to only 9999. This behavior can cause data truncation errors when COBOL statements execute and might effectively limit the maximum length of variable-length character strings to 9999. Consider using the TRUNC(BIN) compiler option or USAGE COMP-5 to avoid data truncation.



Notes:

[1] You cannot use an intervening REDEFINE at level 49.

[2] You cannot directly reference *var-2* as a host variable.

[3] For fixed-length strings, the *picture-string* must be X(*m*) (or XX, with *m* instances of X), where *m*is up to COBOL's limitation. However, the maximum length of the VARCHAR data type in Db2 varies depending on the data page size.

Graphic character host variables

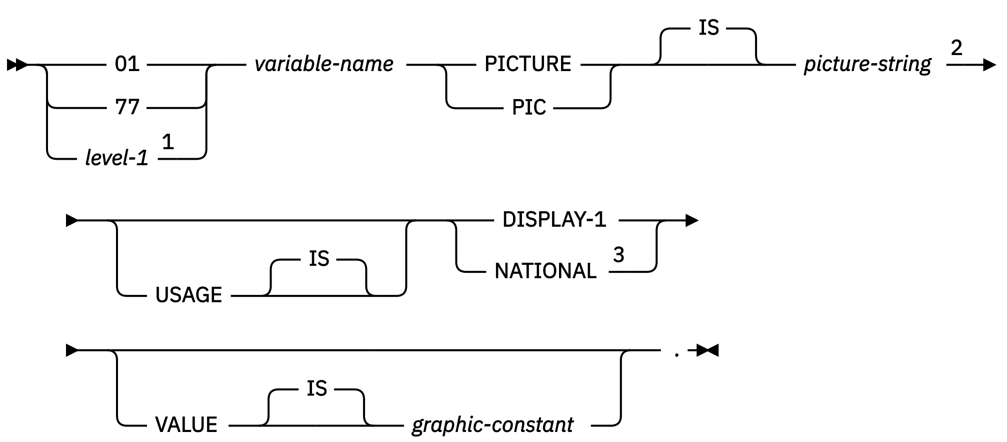
You can specify the following forms of graphic host variables:

- Fixed-length strings
- Varying-length strings

- DBCLOBs

The following diagrams show the syntax for forms other than DBCLOBs.

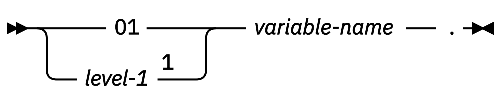
The following diagram shows the syntax for declaring fixed-length graphic host variables.



Notes:

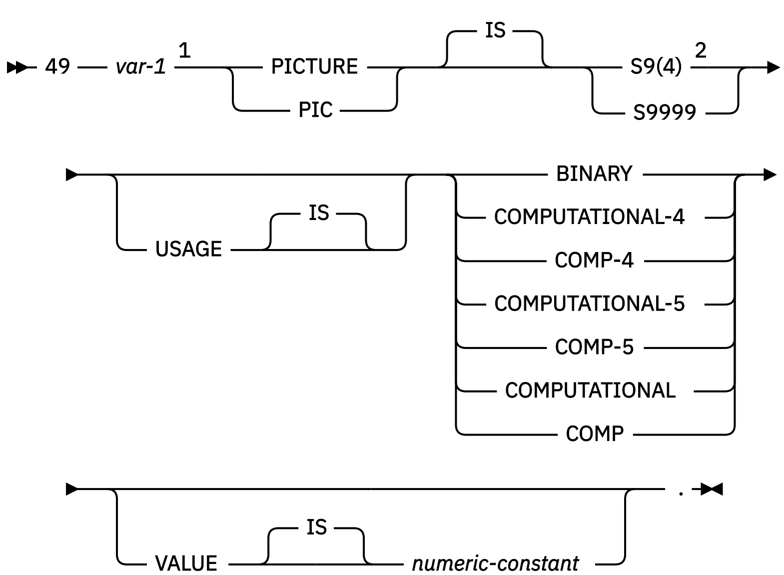
1 *level-1* indicates a COBOL level between 2 and 48.
2 For fixed-length strings, the *picture-string* is G(*m*) or N(*m*) (or, *m* instances of GG...G or NN...N), where *m* is up to COBOL's limitation. However, the maximum length of the GRAPHIC data type (fixed-length graphic string) in Db2 is 127 double-bytes.
3 Use USAGE NATIONAL only for Unicode UTF-16 data. In the *picture-string* for USAGE NATIONAL, you must use N in place of G. USAGE NATIONAL is supported only by the Db2 coprocessor.

The following diagrams show the syntax for declaring varying-length graphic host variables.



Notes:

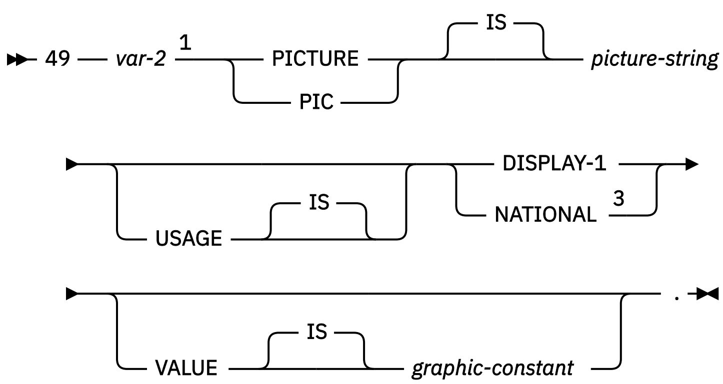
1 *level-1* indicates a COBOL level between 2 and 48.



Notes:

[1] You cannot directly reference *var-1* as a host variable.

[2] Db2 uses the full length of the S9(4) BINARY variable even though COBOL with TRUNC(STD) recognizes values up to only 9999. This behavior can cause data truncation errors when COBOL statements execute and might effectively limit the maximum length of variable-length character strings to 9999. Consider using the TRUNC(BIN) compiler option or USAGE COMP-5 to avoid data truncation.



Notes:

[1] You cannot directly reference *var-2* as a host variable.

[2] For fixed-length strings, the *picture-string* is G(*m*) or N(*m*) (or, *m* instances of GG...G or NN...N), where *m* is up to COBOL's limitation. However, the maximum length of the VARGRAPHIC data type in Db2 varies depending on the data page size.

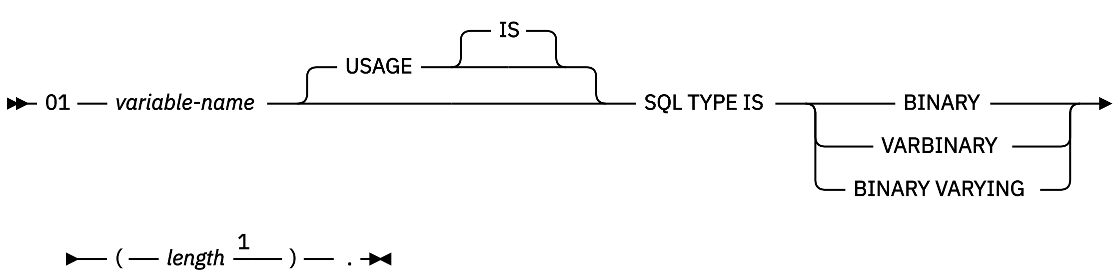
[3] Use USAGE NATIONAL only for Unicode UTF-16 data. In the *picture-string* for USAGE NATIONAL, you must use N in place of G. USAGE NATIONAL is supported only by the Db2 coprocessor.

Binary host variables

You can specify the following forms of binary host variables:

- Fixed-length strings
- Varying-length strings
- BLOBs

The following diagram shows the syntax for declaring BINARY and VARBINARY host variables.



Notes:

[1] For BINARY host variables, the length must be in the range 1 - 255. For VARBINARY host variables, the length must be in the range1 - 32704.

COBOL does not have variables that correspond to the SQL binary types BINARY and VARBINARY. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a COBOL language structure in the output source member.

When you reference a BINARY or VARBINARY host variable in an SQL statement, you must use the variable that you specify in the SQL TYPE declaration. When you reference the host variable in a host language statement, you must use the variable that Db2 generates.

Examples of binary variable declarations

The following table shows examples of variables that Db2 generates when you declare binary host variables.

Table 112. Examples of BINARY and VARBINARY variable declarations for COBOL

Variable declaration that you include in your COBOL program	Corresponding variable that Db2 generates in the output source member
`01 BIN-VAR USAGE IS SQL TYPE IS BINARY(10).`	`01 BIN-VAR PIC X(10).`

`01 VBIN-VAR USAGE IS SQL TYPE IS VARBINARY(10).`	`01 VBIN-VAR.`
	`  49 VBIN-VAR-LEN PIC S9(4) USAGE BINARY.`
	`  49 VBIN-VAR-TEXT PIC X(10).`

Result set locators

The following diagram shows the syntax for declaring result set locators.

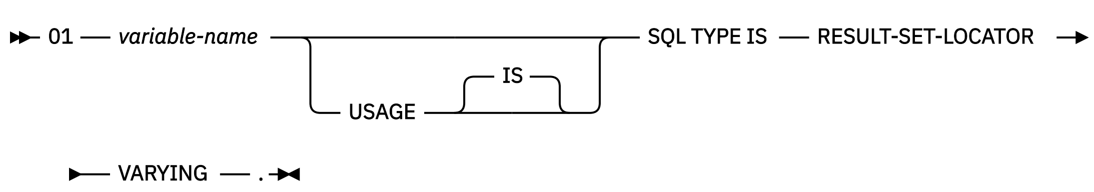
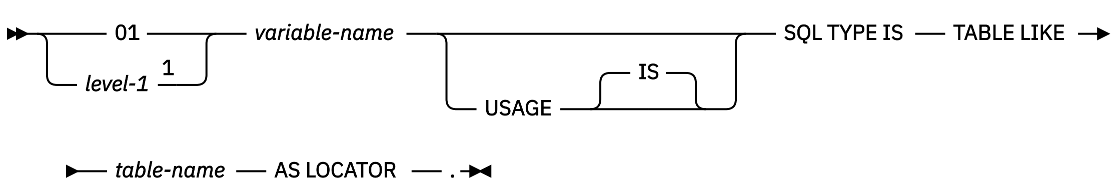


Table Locators

The following diagram shows the syntax for declaring table locators.

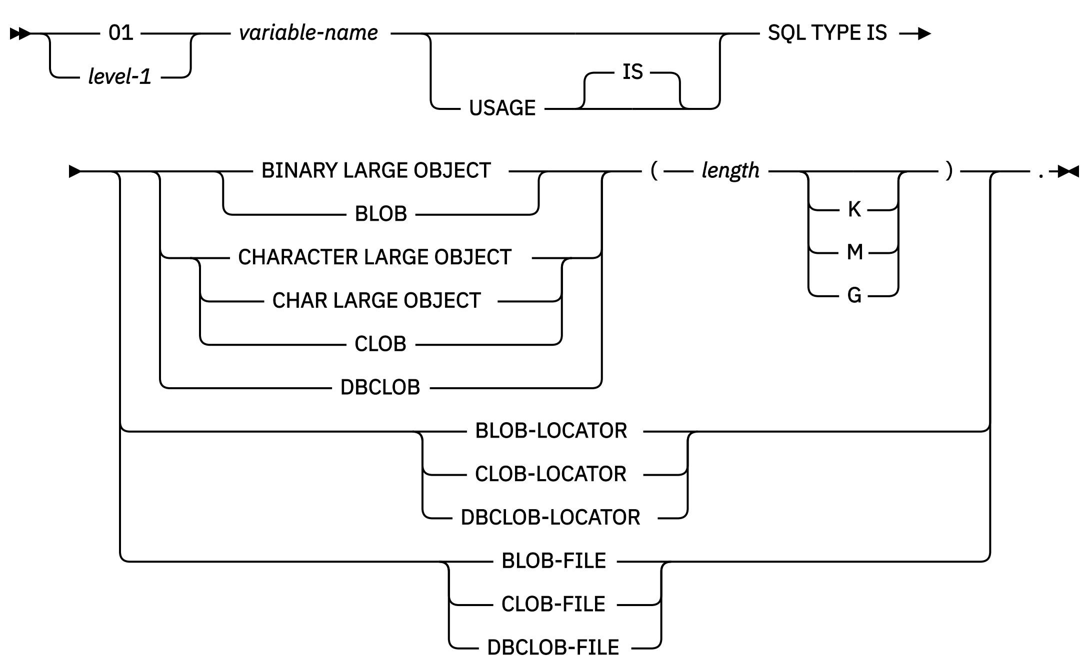


Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

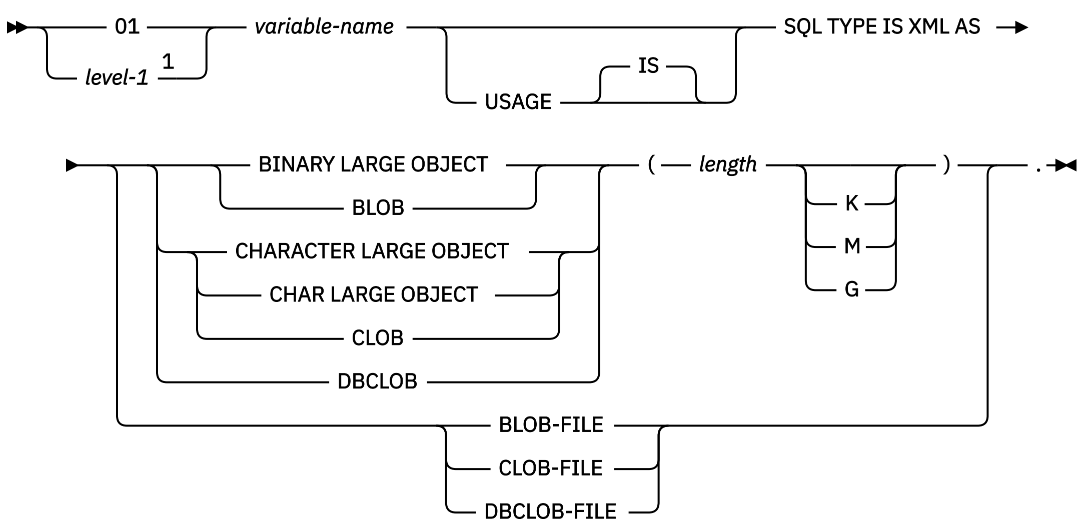
LOB variables and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB variables and file reference variables.



XML data host and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables and file reference variables for XML data types.

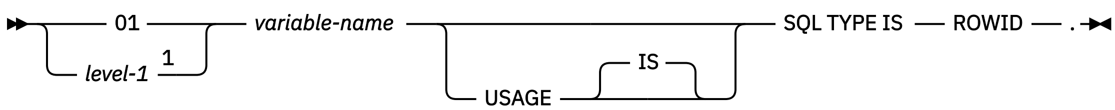


Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

ROWID host variables

The following diagram shows the syntax for declaring ROWID host variables.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Related tasks

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Related reference

Limits in Db2 for z/OS (Db2 SQL)

Host-variable arrays in COBOL

In COBOL programs, you can specify numeric, character, graphic, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host-variable arrays can be referenced only as a simple reference in the following contexts. In syntax diagrams, *host-variable-array* designates a reference to a host-variable array.

- In a FETCH statement for a multiple-row fetch. See FETCH (Db2 SQL).
- In an INSERT statement with multiple rows of source data. This is also referred to as multiple-row insert. See INSERT (Db2 SQL).
- In a MERGE statement with multiple rows of source data. See MERGE (Db2 SQL).
- In an EXECUTE statement to provide a value for a parameter marker in a dynamic multi-row INSERT or MERGE. See EXECUTE (Db2 SQL).

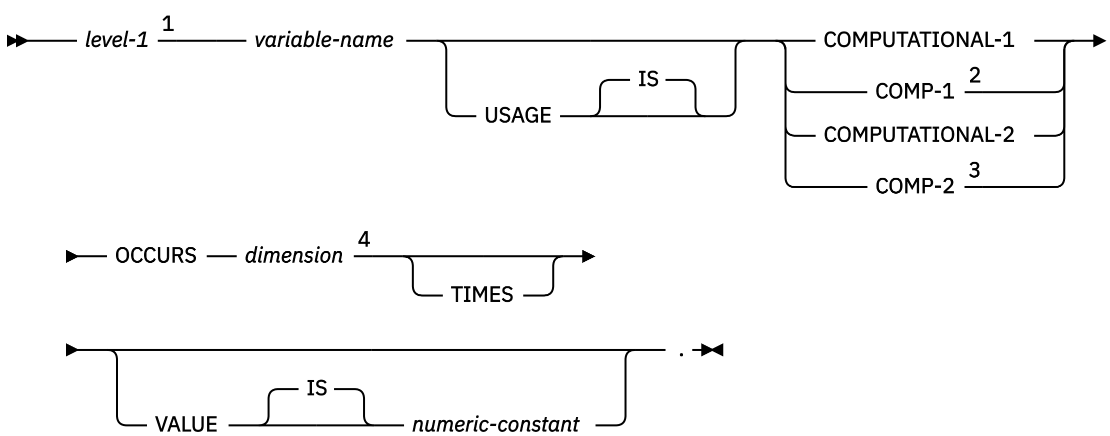
Restriction: Only some of the valid COBOL declarations are valid host-variable array declarations. If the declaration for a variable array is not valid, any SQL statement that references the variable array might result in the message UNDECLARED HOST VARIABLE ARRAY.

Numeric host-variable arrays

You can specify the following forms of numeric host-variable arrays:

- Floating-point numbers
- Integers and small integers
- Decimal numbers

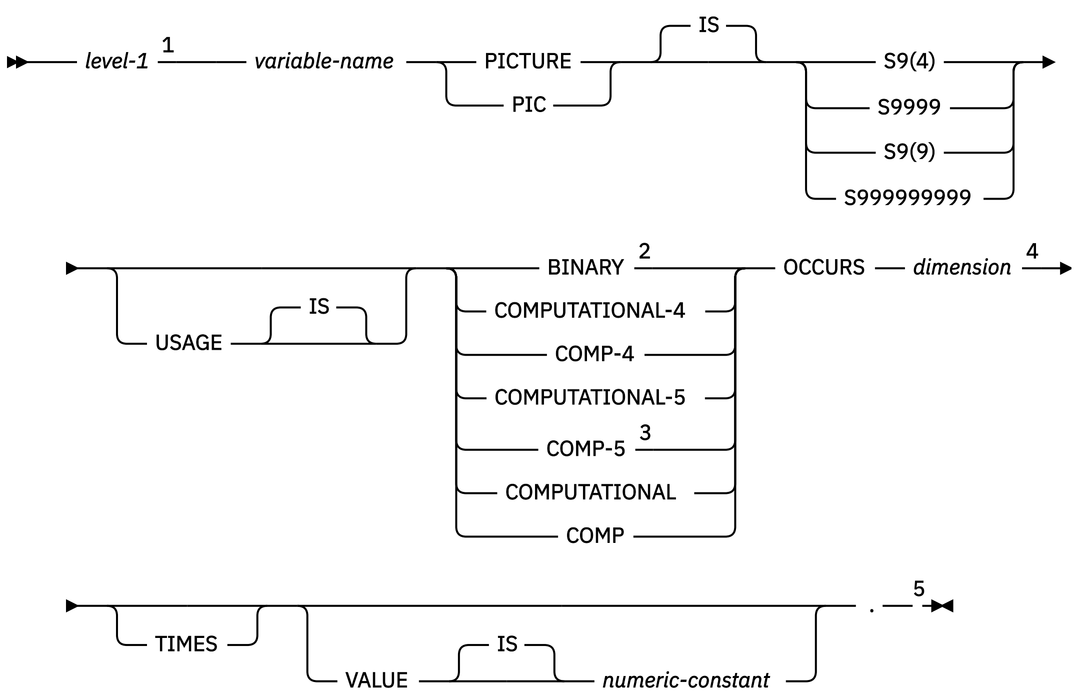
The following diagram shows the syntax for declaring floating-point host-variable arrays.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] COMPUTATIONAL-1 and COMP-1 are equivalent.
[3] COMPUTATIONAL-2 and COMP-2 are equivalent.
[4] *dimension* must be an integer constant between 1 and 32767.

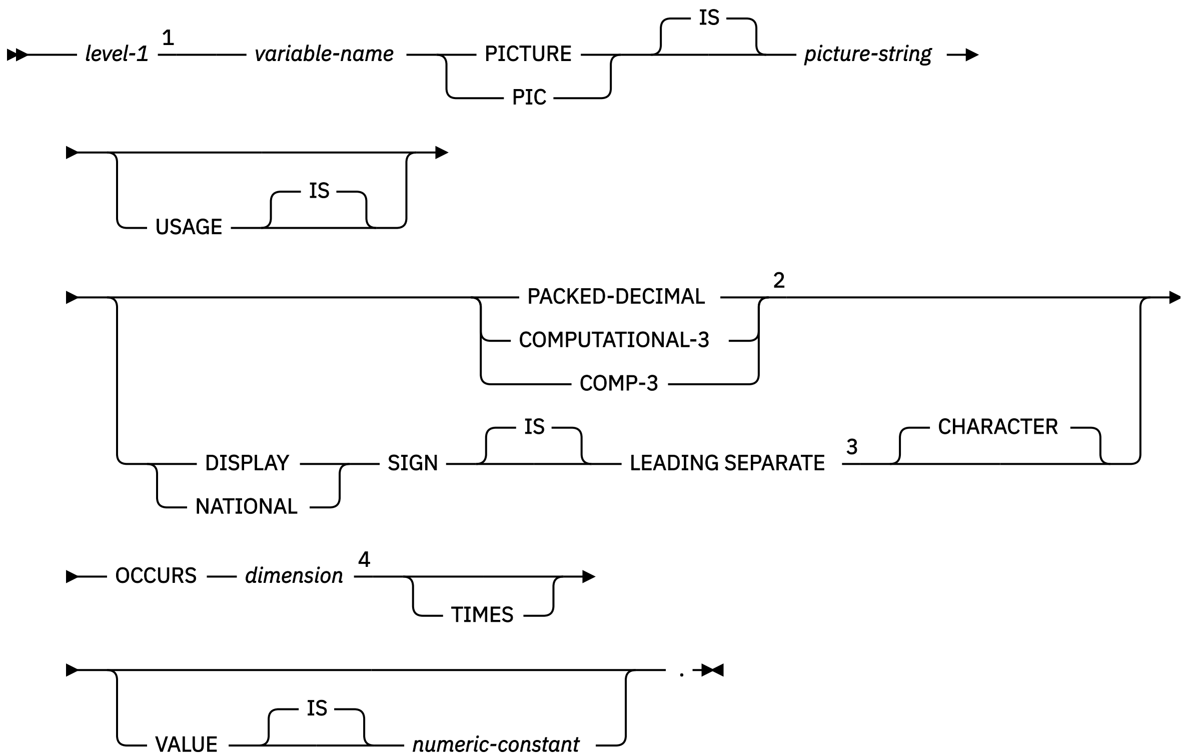
The following diagram shows the syntax for declaring integer and small integer host-variable arrays.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] The COBOL binary integer data types BINARY, COMPUTATIONAL, COMP, COMPUTATIONAL-4, and COMP-4 are equivalent.
[3] COMPUTATIONAL-5 (and COMP-5) are equivalent to the other COBOL binary integer data types if you compile the other data types with TRUNC(BIN).
[4] *dimension* must be an integer constant between 1 and 32767.
[5] Any specification for scale is ignored.

The following diagram shows the syntax for declaring decimal host-variable arrays.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

[2] PACKED-DECIMAL, COMPUTATIONAL-3, and COMP-3 are equivalent. The *picture-string* that is associated with these types must have the form S9(*i*)V9(*d*) (or S9...9V9...9, with *i* and *d* instances of 9) or S9(*i*)V.

[3] The *picture-string* that is associated with SIGN LEADING SEPARATE must have the form S9(*i*)V9(*d*) (or S9...9V9...9, with *i* and *d* instances of 9 or S9...9V with *i* instances of 9).

[4] *dimension* must be an integer constant between 1 and 32767.

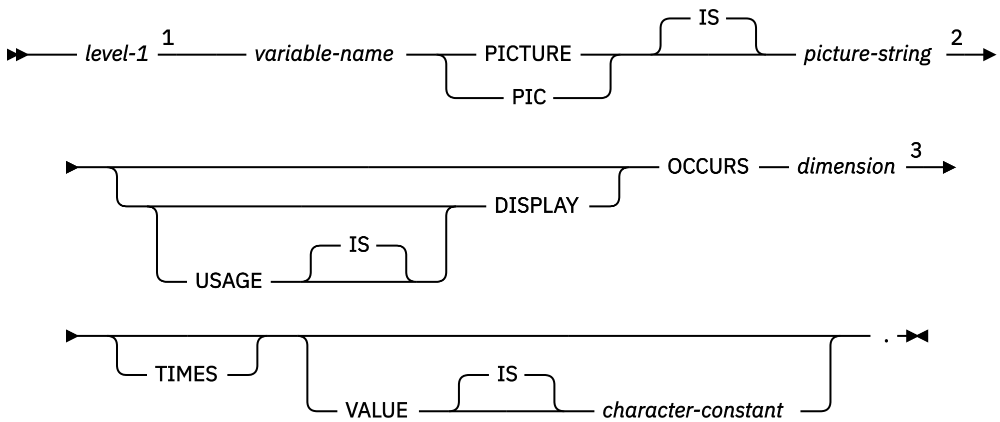
Character host-variable arrays

You can specify the following forms of character host-variable arrays:

- Fixed-length character strings
- Varying-length character strings
- CLOBs

The following diagrams show the syntax for forms other than CLOBs.

The following diagram shows the syntax for declaring fixed-length character string arrays.



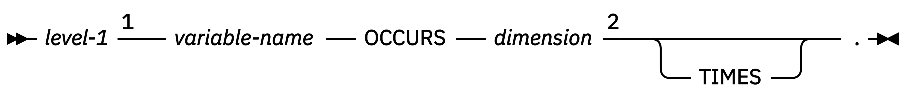
Notes:

1 *level-1* indicates a COBOL level between 2 and 48.

2 The *picture-string* must be in the form X(*m*) (or XX...X, with *m* instances of X), where 1 <= *m* <= 32767 for fixed-length strings. However, the maximum length of the CHAR data type (fixed-length character string) in Db2 is 255 bytes.

3 *dimension* must be an integer constant between 1 and 32767.

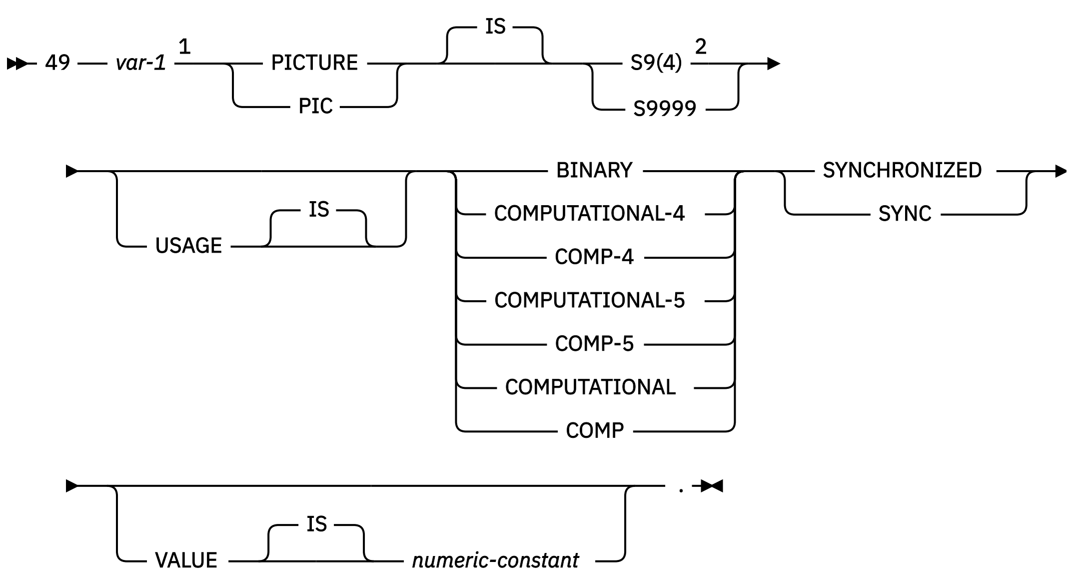
The following diagrams show the syntax for declaring varying-length character string arrays.



Notes:

1 *level-1* indicates a COBOL level between 2 and 48.

2 *dimension* must be an integer constant between 1 and 32767.

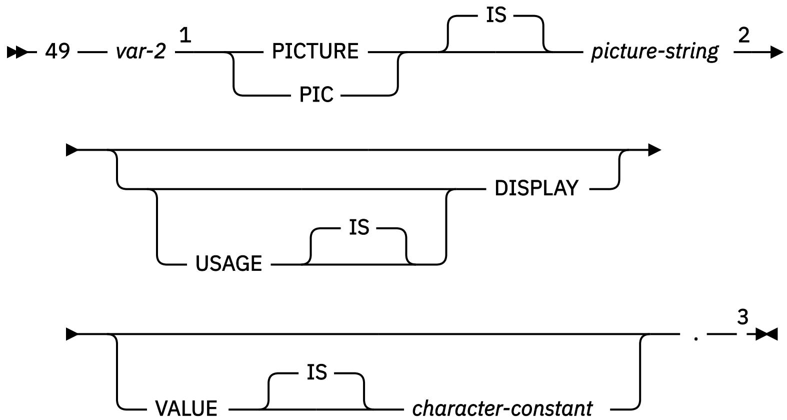


Notes:

1 You cannot directly reference *var-1* as a host-variable array.

2 Db2 uses the full length of the S9(4) BINARY variable even though COBOL with TRUNC(STD) recognizes values up to only 9999. This behavior can cause data truncation errors when COBOL statements execute

and might effectively limit the maximum length of variable-length character strings to 9999. Consider using the TRUNC(BIN) compiler option or USAGE COMP-5 to avoid data truncation.



Notes:

[1] You cannot directly reference *var-2* as a host-variable array.

[2] The *picture-string* must be in the form X(*m*) (or XX...X, with *m* instances of X), where 1 <= *m* <= 32767 for fixed-length strings; for other strings, *m* cannot be greater than the maximum size of a varying-length character string.

[3] You cannot use an intervening REDEFINE at level 49.

The following example shows declarations of a fixed-length character array and a varying-length character array.

```
01  OUTPUT-VARS.
    05  NAME OCCURS 10 TIMES.
        49 NAME-LEN   PIC S9(4) COMP-4 SYNC.
        49 NAME-DATA  PIC X(40).
    05  SERIAL-NUMBER PIC S9(9) COMP-4 OCCURS 10 TIMES.
```

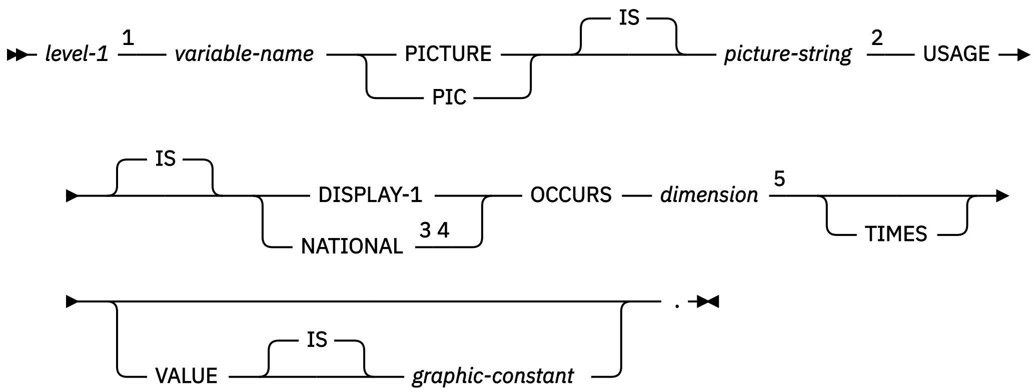
Graphic character host-variable arrays

You can specify the following forms of graphic host-variable arrays:

- Fixed-length strings
- Varying-length strings
- DBCLOBs

The following diagrams show the syntax for forms other than DBCLOBs.

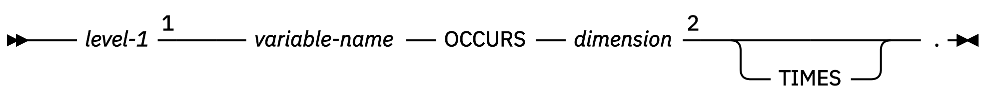
The following diagram shows the syntax for declaring fixed-length graphic string arrays.



Notes:

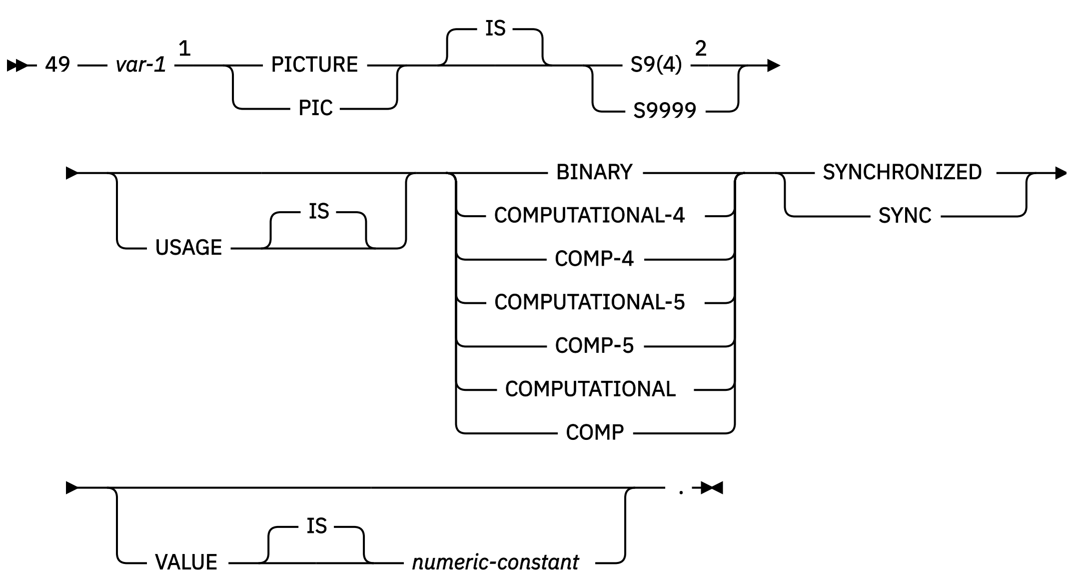
[1] *level-1* indicates a COBOL level between 2 and 48.
[2] For fixed-length strings, the format for *picture-string* is G(*m*) or N(*m*) (or, *m* instances of GG...G or NN...N), where 1 <= *m* <= 127; for other strings, *m* cannot be greater than the maximum size of a varying-length graphic string.
[3] Use USAGE NATIONAL only for Unicode UTF-16 data. In the *picture-string* for USAGE NATIONAL, you must use N in place of G.
[4] You can use USAGE NATIONAL only if you are using the Db2 coprocessor.
[5] *dimension* must be an integer constant between 1 and 32767.

The following diagrams show the syntax for declaring varying-length graphic string arrays.



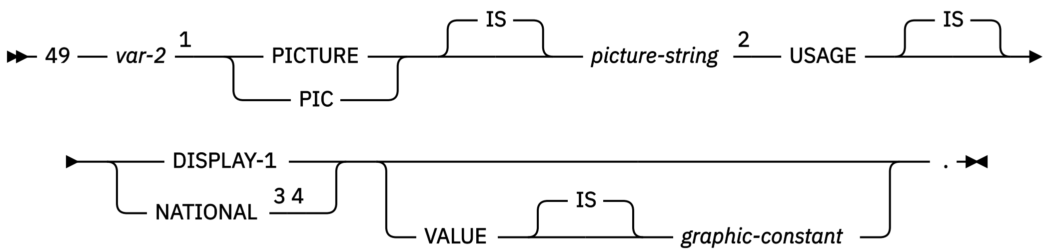
Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] *dimension* must be an integer constant between 1 and 32767.



Notes:

[1] You cannot directly reference *var-1* as a host-variable array.
[2] Db2 uses the full length of the S9(4) BINARY variable even though COBOL with TRUNC(STD) recognizes values up to only 9999. This behavior can cause data truncation errors when COBOL statements execute and might effectively limit the maximum length of variable-length character strings to 9999. Consider using the TRUNC(BIN) compiler option or USAGE COMP-5 to avoid data truncation.

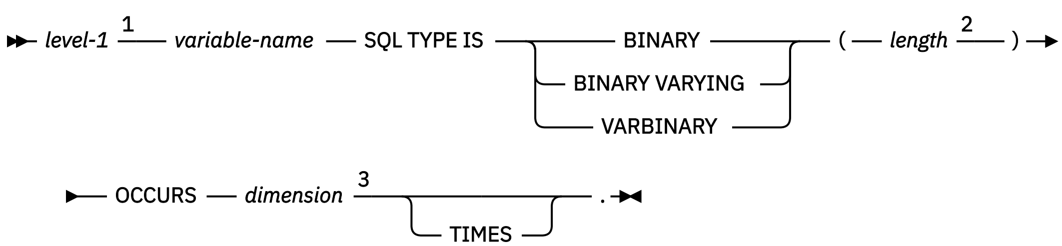


Notes:

[1] You cannot directly reference *var-2* as a host-variable array.
[2] For fixed-length strings, the format for *picture-string* is G(*m*) or N(*m*) (or, *m* instances of GG...G or NN...N), where 1 <= *m* <= 127; for other strings, *m* cannot be greater than the maximum size of a varying-length graphic string.
[3] Use USAGE NATIONAL only for Unicode UTF-16 data. In the *picture-string* for USAGE NATIONAL, you must use N in place of G.
[4] You can use USAGE NATIONAL only if you are using the Db2 coprocessor.

Binary host-variable arrays

The following diagram shows the syntax for declaring binary host-variable arrays.

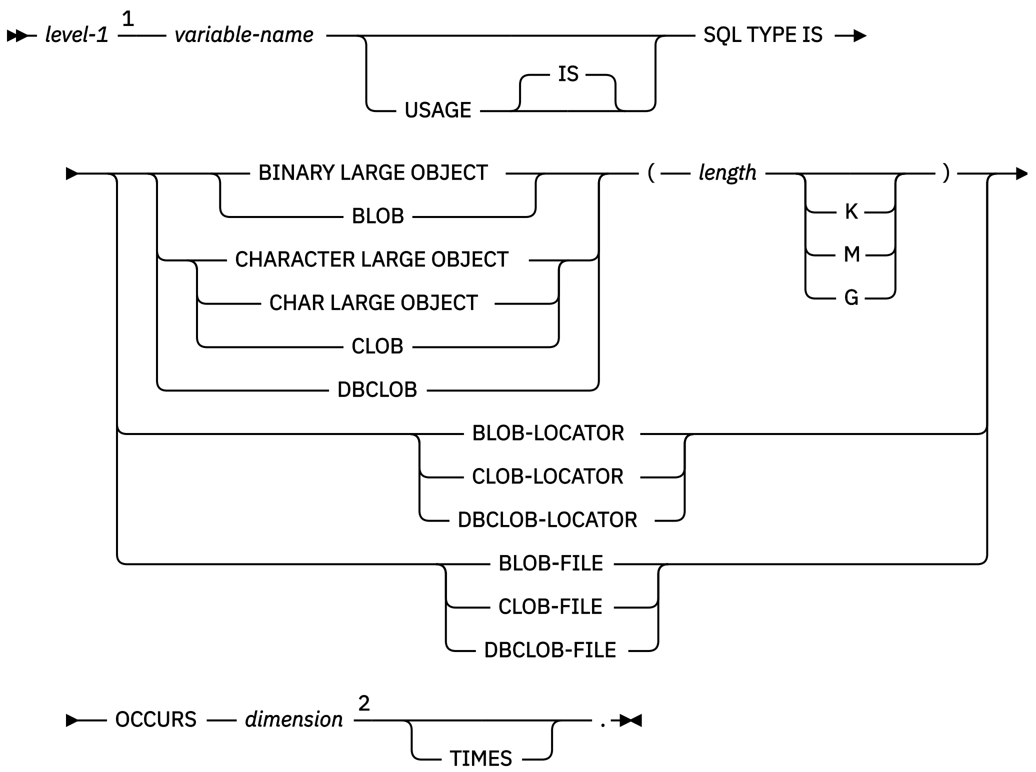


Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] For BINARY host variables, the *length* must be in the range 1 to 255. For VARBINARY host variables, the *length* must be in the range 1 to 32704.
[3] *dimension* must be an integer constant between 1 and 32767.

LOB, locator, and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variable, locator, and file reference arrays.



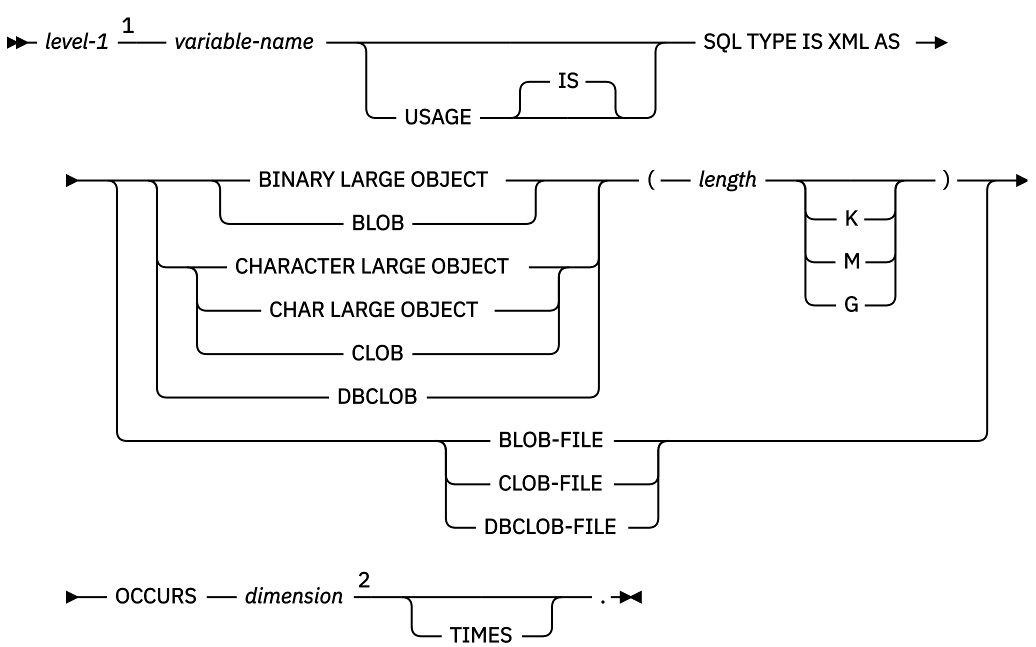
Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.

[2] *dimension* must be an integer constant between 1 and 32767.

XML host and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variable and file reference arrays for XML data types.

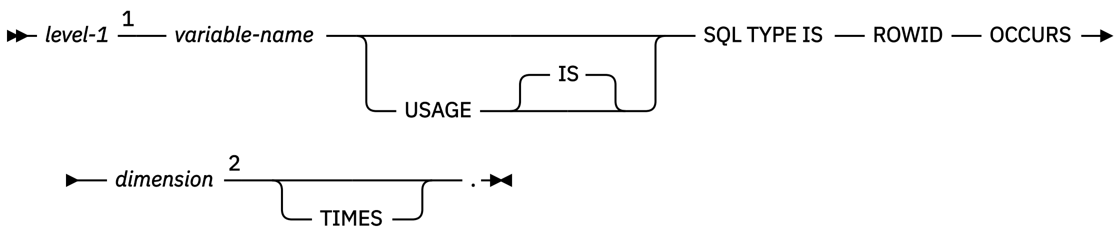


Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] *dimension* must be an integer constant between 1 and 32767.

ROWID variable arrays

The following diagram shows the syntax for declaring ROWID variable arrays.



Notes:

[1] *level-1* indicates a COBOL level between 2 and 48.
[2] *dimension* must be an integer constant between 1 and 32767.

Related concepts

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

Host-variable arrays
You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Host structures in COBOL

A COBOL host structure is a named set of host variables that are defined in your program's WORKING-STORAGE SECTION or LINKAGE SECTION.

Requirements: Host structure declarations in COBOL must satisfy the following requirements:

- COBOL host structures can have a maximum of two levels, even though the host structure might occur within a structure with multiple levels. However, you can declare a varying-length character string, which must be level 49.
- A host structure name can be a group name whose subordinate levels name elementary data items.
- If you are using the Db2 precompiler, do not declare host variables or host structures on any subordinate levels after one of the following items:
  - A COBOL item that begins in area A

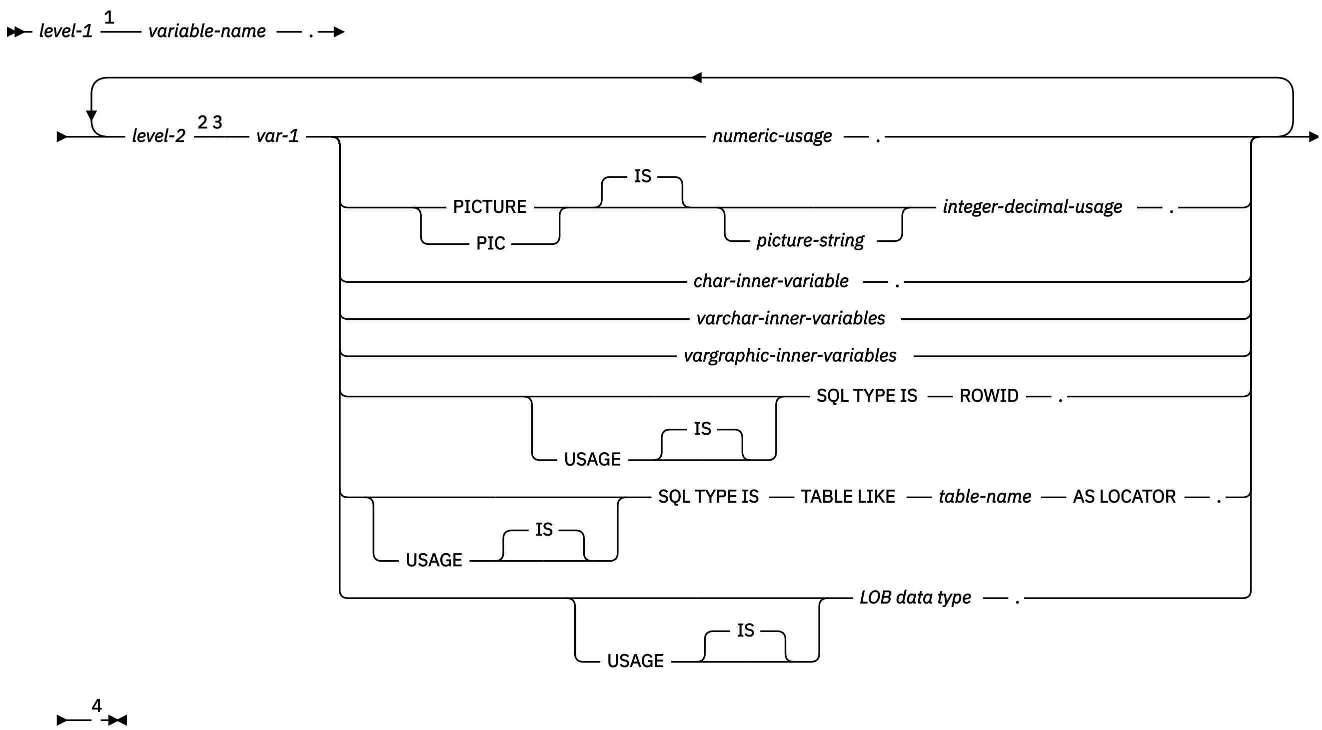
– Any SQL statement (except SQL INCLUDE)
– Any SQL statement within an included member

When the Db2 precompiler encounters one of the preceding items in a host structure, it considers the structure to be complete.

When you write an SQL statement that contains a qualified host variable name (perhaps to identify a field within a structure), use the name of the structure followed by a period and the name of the field. For example, for structure B that contains field C1, specify `B.C1` rather than `C1 OF B` or `C1 IN B`.

Host structures

The following diagram shows the syntax for declaring host structures.

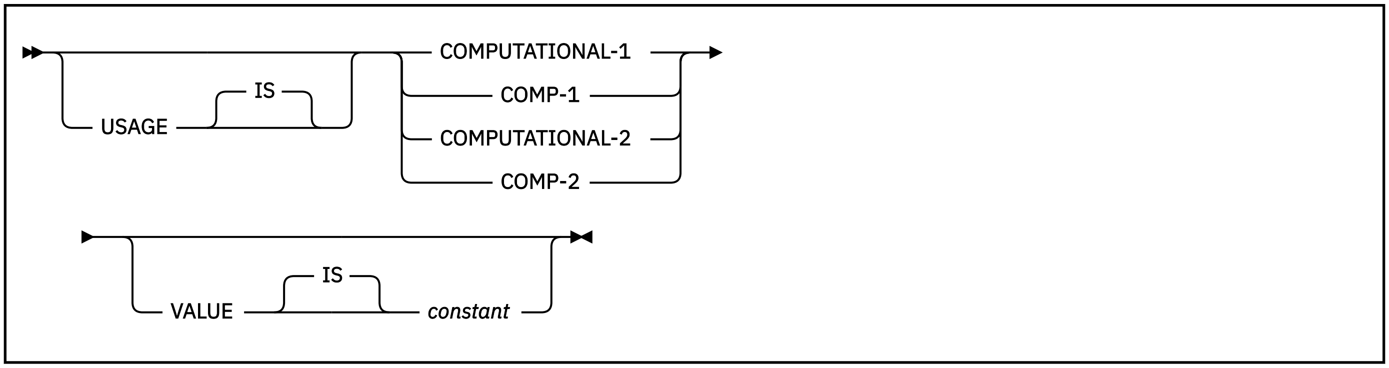


Notes:

[1] *level-1* indicates a COBOL level between 1 and 47.
[2] *level-2* indicates a COBOL level between 2 and 48.
[3] For elements within a structure, use any level 02 through 48 (rather than 01 or 77), up to a maximum of two levels.
[4] Using a FILLER or optional FILLER item within a host structure declaration can invalidate the whole structure.

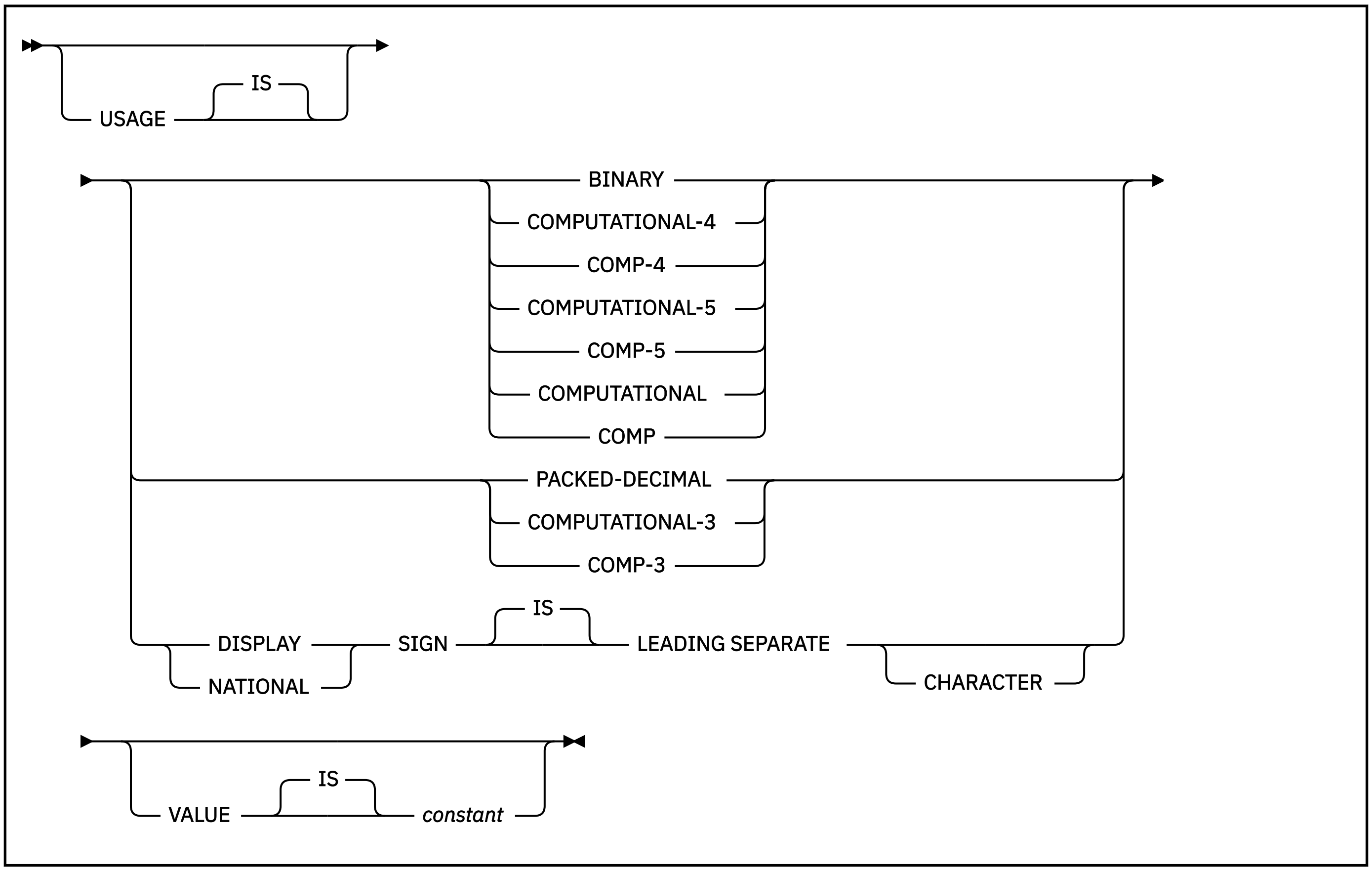
Numeric usage items

The following diagram shows the syntax for numeric-usage items that are used within declarations of host structures.



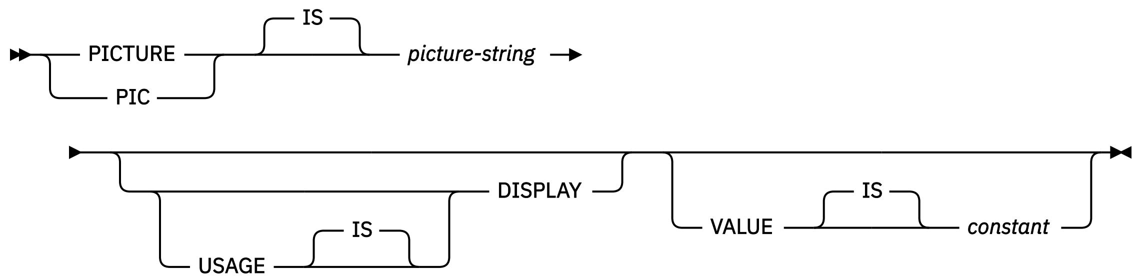
Integer and decimal usage items

The following diagram shows the syntax for integer and decimal usage items that are used within declarations of host structures.



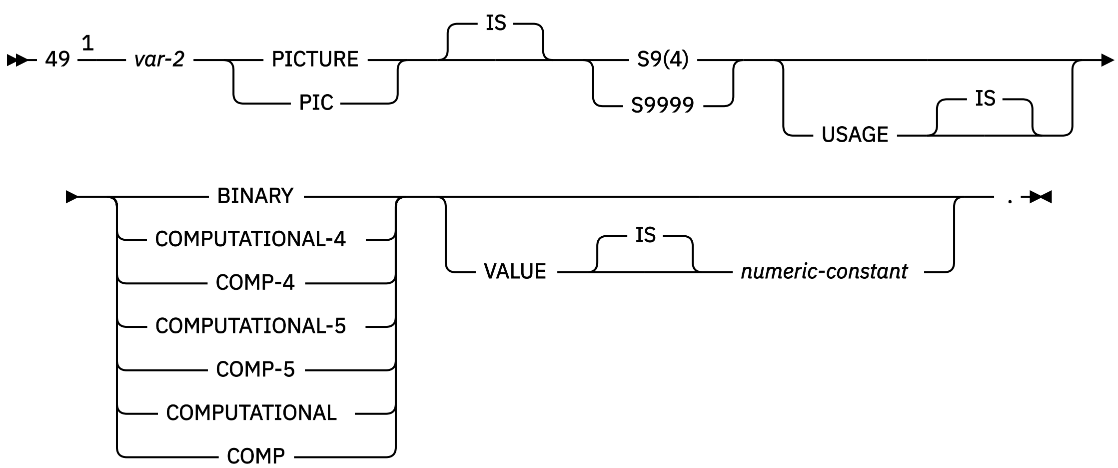
CHAR inner variables

The following diagram shows the syntax for CHAR inner variables that are used within declarations of host structures.



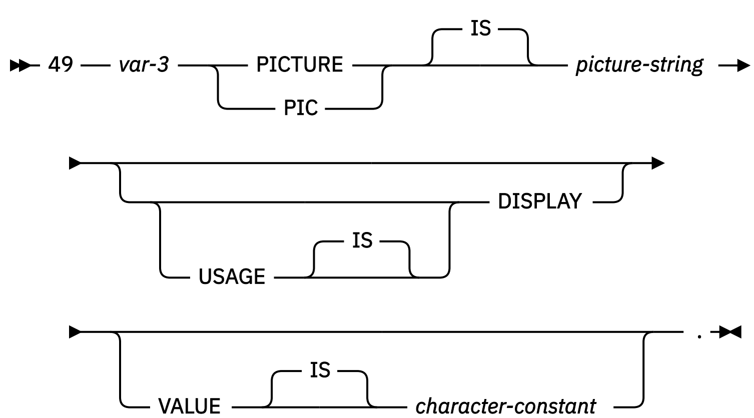
VARCHAR inner variables

The following diagrams show the syntax for VARCHAR inner variables that are used within declarations of host structures.



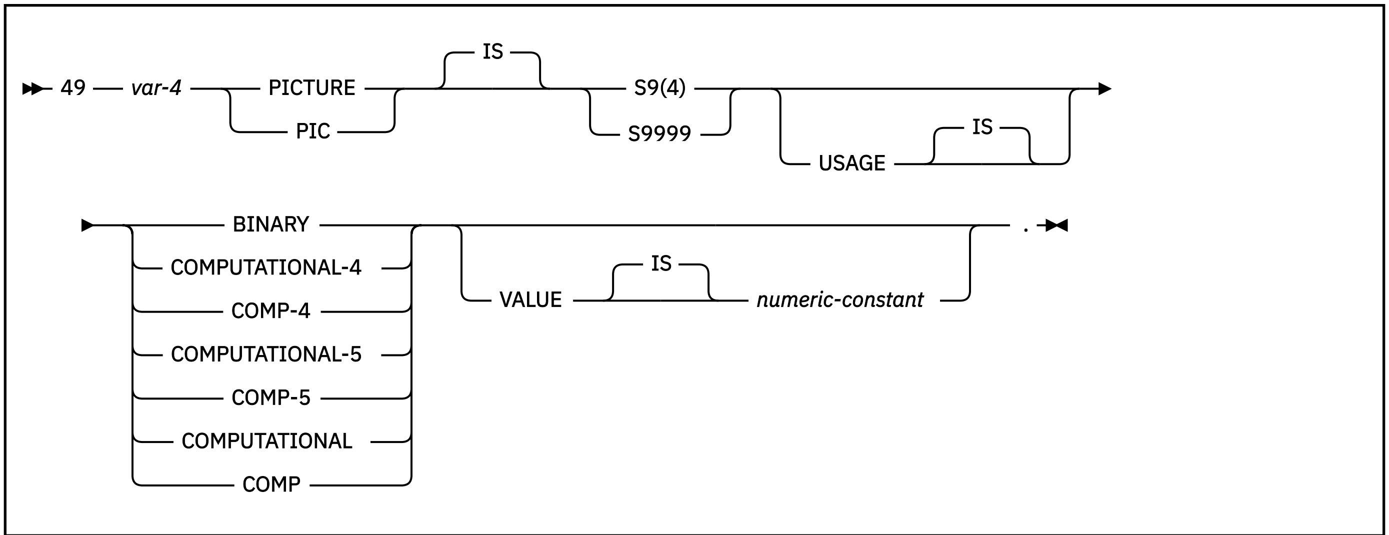
Notes:

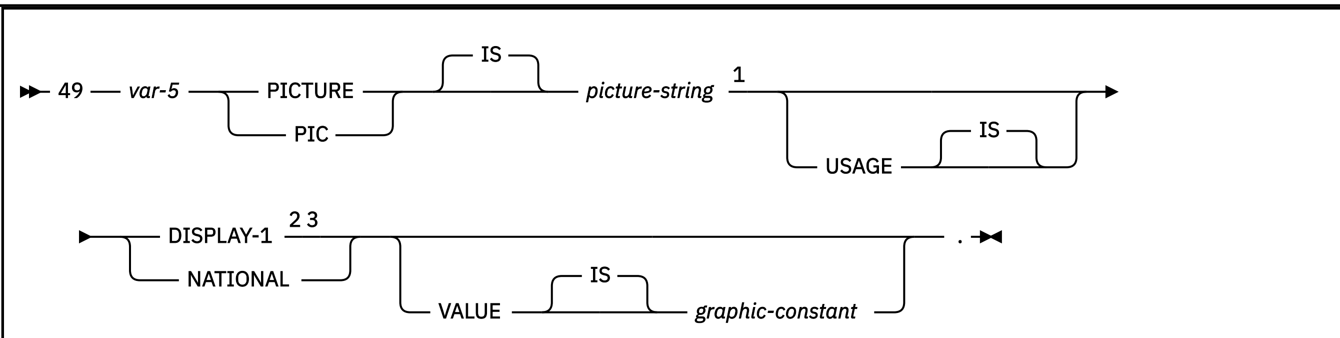
[1] The number 49 has a special meaning to Db2. Do not specify another number.



VARGRAPHIC inner variables

The following diagrams show the syntax for VARGRAPHIC inner variables that are used within declarations of host structures.





Notes:

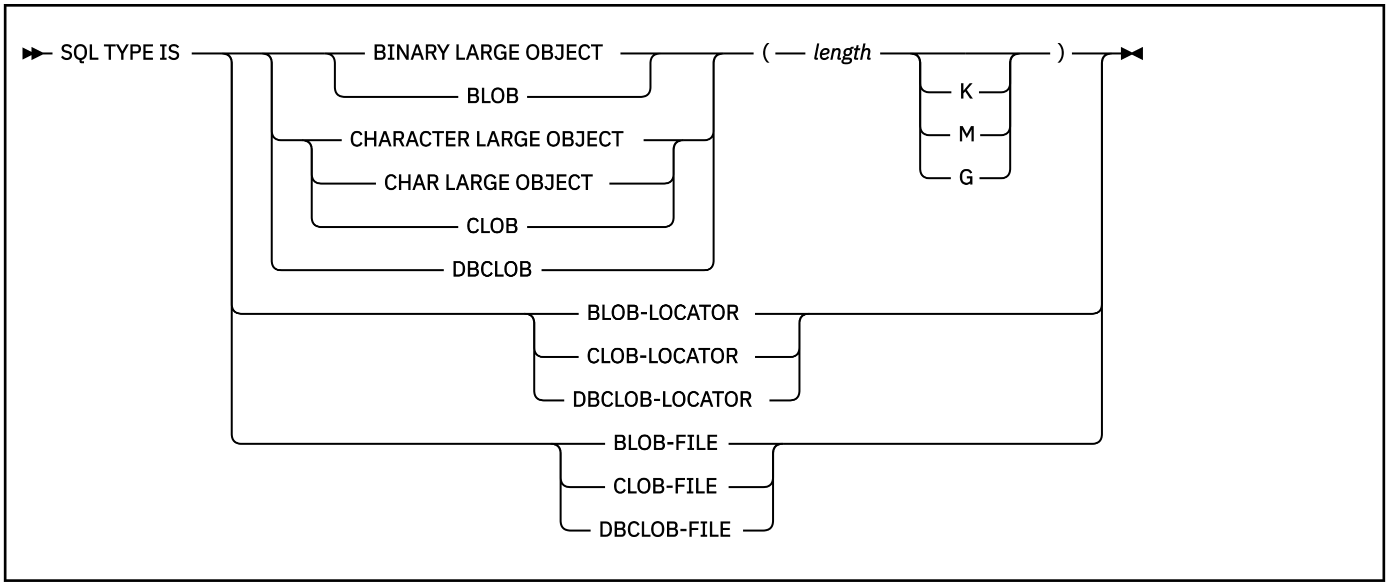
[1] For fixed-length strings, the format of *picture-string* is G(*m*) or N(*m*) (or, *m* instances of GG...G or NN...N), where 1 <= *m* <= 127; for other strings, *m* cannot be greater than the maximum size of a varying-length graphic string.

[2] Use USAGE NATIONAL for only Unicode UTF-16 data. In the *picture-string* for USAGE NATIONAL, you must use N in place of G.

[3] You can use USAGE NATIONAL only if you are using the Db2 coprocessor.

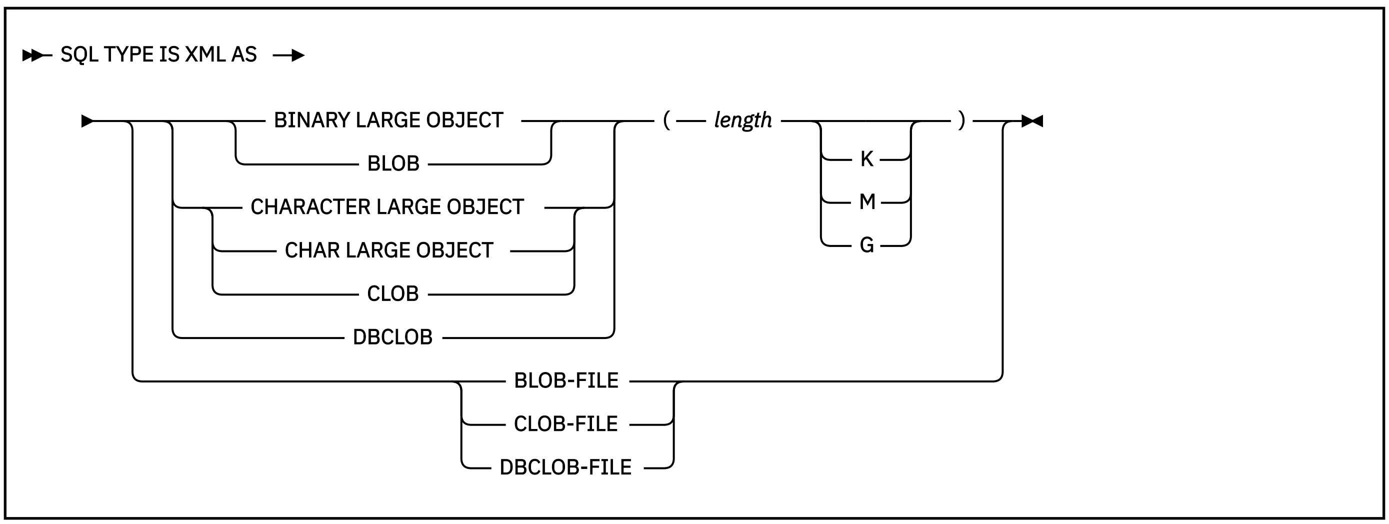
LOB variables, locators, and file reference variables

The following diagram shows the syntax for LOB variables, locators, and file reference variables that are used within declarations of host structures.



LOB variables and file reference variables for XML data

The following diagram shows the syntax for LOB variables and file reference variables that are used within declarations of host structures for XML.



Example

In the following example, B is the name of a host structure that contains the elementary items C1 and C2.

```
01 A
  02 B
    03 C1 PICTURE ...
    03 C2 PICTURE ...
```

To reference the C1 field in an SQL statement, specify `B.C1`.

Related concepts

Host structures

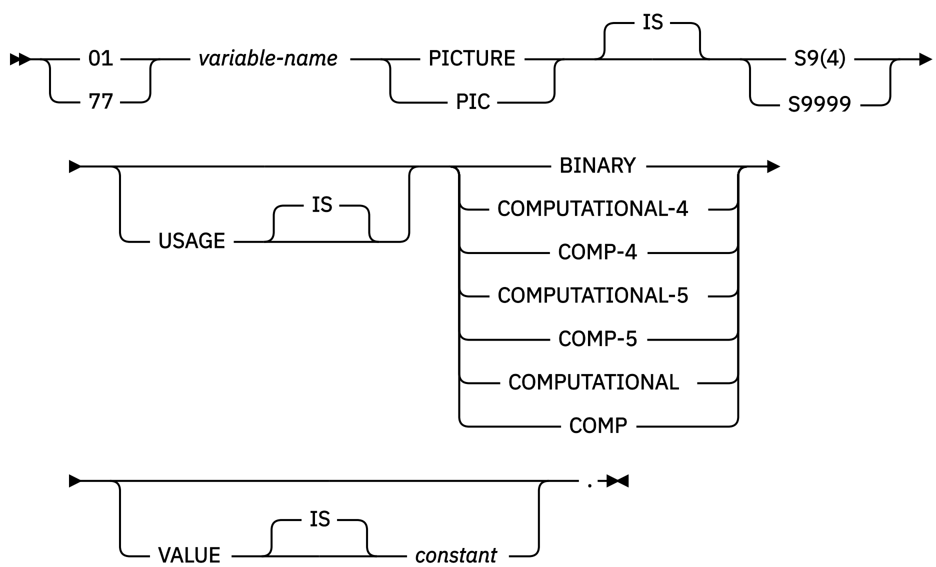
Use host structures to pass a group of host variables between Db2 and your application.

Indicator variables, indicator arrays, and host structure indicator arrays in COBOL

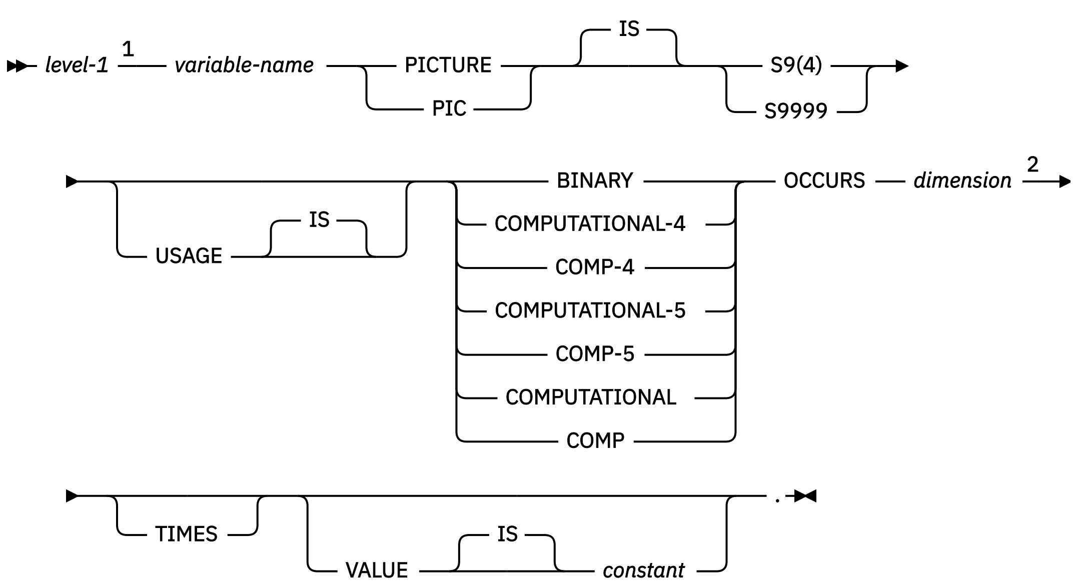
A COBOL indicator variable is a 2-byte binary integer. A COBOL indicator variable array is an array in which each element is declared as a 2-byte binary integer. You can use indicator variable arrays to support COBOL host structures.

You declare indicator variables in the same way that you declare host variables.

The following diagram shows the syntax for declaring an indicator variable in COBOL.



The following diagram shows the syntax for declaring an indicator array in COBOL.



Notes:

[1] *level-1* must be an integer between 2 and 48.

[2] *dimension* must be an integer constant between 1 and 32767.

Examples

Example 1:

The following example shows declarations of three variables, with an indicator variable declaration for each host variable.

```
77 DAYVAR     PIC S9(4) BINARY.
77 BGNVAR     PIC X(8).
77 ENDVAR     PIC X(8).
77 DAYVAR-IND PIC S9(4) BINARY.
77 BGNVAR-IND PIC S9(4) BINARY.
77 ENDVAR-IND PIC S9(4) BINARY.
```

The following FETCH statement retrieves values from a table into the three host variables. If a table column value is null, Db2 sets the corresponding indicator variable to -1.

```
EXEC SQL FETCH CLS_CURSOR INTO
                               :DAYVAR:DAYVAR-IND,
                               :BGNVAR:BGNVAR-IND,
                               :ENDVAR:ENDVAR-IND
END-EXEC.
```

Example 2:

The following example shows a declaration of a host structure, and a corresponding structure that contains an indicator array with the same number of elements as the number of variables in the host structure.

```
01 CLS.
   10 DAYVAR    PIC S9(4) BINARY.
   10 BGNVAR    PIC X(8).
   10 ENDVAR    PIC X(8).
01 CLS-IND-STRUCT.
   02 CLS-IND   PIC S9(4) BINARY OCCURS 3 TIMES.
```

The following FETCH statement retrieves values from a table into the host structure. If a table value is null, Db2 sets the corresponding indicator array element to -1.

```
EXEC SQL FETCH CLS_CURSOR INTO
  :CLS:CLS-IND
END-EXEC.
```

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related tasks

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Controlling the CCSID for COBOL host variables

Setting the CCSID for COBOL host variables is slightly different than the process for other host languages. In COBOL, several other settings affect the CCSID.

Before you begin

This task applies to programs that use IBM Enterprise COBOL for z/OS and the Db2 coprocessor.

Procedure

Use one or more of the following items:

The NATIONAL data type
Use this data type to declare Unicode values in the UTF-16 format (CCSID 1200).

If you declare a host variable HV1 as USAGE NATIONAL, Db2 always handles HV1 as if you had used the following DECLARE VARIABLE statement:

```
DECLARE :HV1 VARIABLE CCSID 1200
```

The COBOL CODEPAGE compiler option
Use this option to specify the default EBCDIC CCSID of character data items.

The SQLCCSID compiler option
Use this option to control whether the CODEPAGE compiler option influences the processing of SQL host variables in your COBOL programs (available in Enterprise COBOL V3R4 or later).

When you specify the SQLCCSID compiler option, the COBOL Db2 coprocessor uses the CCSID that is specified in the CODEPAGE compiler option. All host variables of character data type, other than NATIONAL, are specified with that CCSID unless they are explicitly overridden by a DECLARE VARIABLE statement.

When you specify the NOSQLCCSID compiler option, the CCSID that is specified in the CODEPAGE compiler option is used for processing only COBOL statements within the COBOL program. That CCSID is not used for the processing of host variables in SQL statements. Db2 uses the CCSIDs that are specified through Db2 mechanisms and defaults as host variable data value encodings.

The DECLARE VARIABLE statement.
This statement explicitly sets the CCSID for individual host variables.

Example

Assume that the COBOL SQLCCSID compiler option is specified and that the COBOL CODEPAGE compiler option is specified as CODEPAGE(1141). The following code shows how you can control the CCSID:

```
DATA DIVISION.
     01  HV1  PIC N(10) USAGE NATIONAL.
     01  HV2  PIC X(20) USAGE DISPLAY.
     01  HV3  PIC X(30) USAGE DISPLAY.
     ...
     EXEC SQL
       DECLARE :HV3 VARIABLE CCSID 1047
     END-EXEC.
     ...
PROCEDURE DIVISION.
     ...
     EXEC SQL
       SELECT C1, C2, C3 INTO :HV1, :HV2, :HV3 FROM T1
     END-EXEC.
```

Each of the host variables have the following CCSIDs:

HV1
1200

HV2
1141

HV3
1047

Assume that the COBOL NOSQLCCSID compiler option is specified, the COBOL CODEPAGE compiler option is specified as CODEPAGE(1141), and the Db2 default single byte CCSID is set to 37. In this case, each of the host variables in this example have the following CCSIDs:

HV1
1200

HV2
37

HV3
1047

Related reference

Host variables in COBOL
In COBOL programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set and table locators and LOB and XML file reference variables.

Compiler options (COBOL) (Enterprise COBOL for z/OS Programming Guide)

Equivalent SQL and COBOL data types

When you declare host variables in your COBOL programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

The following table describes the SQL data type and the base SQLTYPE and SQLLEN values that the precompiler uses for host variables in SQL statements.

Table 113. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in COBOL programs

COBOL host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
COMP-1	480	4	REAL **or** FLOAT(*n*) 1≤*n*≤21
COMP-2	480	8	DOUBLE PRECISION, **or** FLOAT(*n*) 22≤*n*≤53
S9(*i*)V9(*d*) COMP-3 or S9(*i*)V9(*d*) PACKED-DECIMAL	484	*i*+*d* in byte 1, *d* in byte 2	DECIMAL(*i*+*d*,*d*) **or** NUMERIC(*i*+*d*,*d*)
S9(*i*)V9(*d*) DISPLAY SIGN LEADING SEPARATE	504	*i*+*d* in byte 1, *d* in byte 2	No exact equivalent. Use DECIMAL(*i*+*d*,*d*) **or** NUMERIC(*i*+*d*,*d*)
S9(*i*)V9(*d*) NATIONAL SIGN LEADING SEPARATE	504	*i*+*d* in byte 1, *d* in byte 2	No exact equivalent. Use DECIMAL(*i*+*d*,*d*) **or** NUMERIC(*i*+*d*,*d*)
S9(4) COMP-4, S9(4) COMP-5, S9(4) COMP, or S9(4) BINARY	500	2	SMALLINT
S9(9) COMP-4, S9(9) COMP-5, S9(9) COMP, or S9(9) BINARY	496	4	INTEGER
S9(18) COMP-4, S9(18) COMP-5, S9(18) COMP, or S9(18) BINARY	492	8	BIGINT
Fixed-length character data	452	*n*	CHAR(*n*)
Varying-length character data 1≤*n*≤255	448	*n*	VARCHAR(*n*)
Varying-length character data *m*>255	456	*m*	VARCHAR(*m*)
Fixed-length graphic data	468	*m*	GRAPHIC(*m*)
Varying-length graphic data 1≤*m*≤127	464	*m*	VARGRAPHIC(*m*)

Table 113. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in COBOL programs (continued)

COBOL host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
Varying-length graphic data *m*>127	472	*m*	VARGRAPHIC(*m*)
SQL TYPE is BINARY(*n*), 1≤*n*≤255	912	*n*	BINARY(*n*)
SQL TYPE is VARBINARY(*n*), 1≤*n*≤32704	908	*n*	VARBINARY(*n*)
SQL TYPE IS RESULT-SET-LOCATOR	972	4	Result set locator[2]
SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	976	4	Table locator[2]
SQL TYPE IS BLOB-LOCATOR	960	4	BLOB locator[2]
SQL TYPE IS CLOB-LOCATOR	964	4	CLOB locator[2]
SQL TYPE IS DBCLOB-LOCATOR	968	4	DBCLOB locator[2]
USAGE IS SQL TYPE IS BLOB(*i*) 1≤*i*≤2147483647	404	*i*	BLOB(*i*)
USAGE IS SQL TYPE IS CLOB(*i*) 1≤*i*≤2147483647	408	*i*	CLOB(*i*)
USAGE IS SQL TYPE IS DBCLOB(*m*) 1≤*m*≤1073741823[3]	412	*i*	DBCLOB(*m*)[3]
SQL TYPE IS XML AS BLOB(*i*)	404	0	XML
SQL TYPE IS XML AS CLOB(*i*)	408	0	XML
SQL TYPE IS XML AS DBCLOB(*i*)	412	0	XML
SQL TYPE IS BLOB-FILE	916/917	267	BLOB file reference[2]
SQL TYPE IS CLOB-FILE	920/921	267	CLOB file reference[2]
SQL TYPE IS DBCLOB-FILE	924/925	267	DBCLOB file reference[2]
SQL TYPE IS XML AS BLOB-FILE	916/917	267	XML BLOB file reference[2]
SQL TYPE IS XML AS CLOB-FILE	920/921	267	XML CLOB file reference[2]
SQL TYPE IS XML AS DBCLOB-FILE	924/925	267	XML DBCLOB file reference[2]
SQL TYPE IS ROWID	904	40	ROWID

Table 113. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in COBOL programs (continued)			
COBOL host variable data type	**SQLTYPE of host variable[1]**	**SQLLEN of host variable**	**SQL data type**

Notes:

1. If a host variable includes an indicator variable, the SQLTYPE value is the base SQLTYPE value plus 1.
2. Do not use this data type as a column type.
3. *m* is the number of double-byte characters.

The following table shows equivalent COBOL host variables for each SQL data type. Use this table to determine the COBOL data type for host variables that you define to receive output from the database. For example, if you retrieve TIMESTAMP data, you can define a fixed-length character string variable of length *n*

This table shows direct conversions between SQL data types and COBOL data types. However, a number of SQL data types are compatible. When you do assignments or comparisons of data that have compatible data types, Db2 converts those compatible data types.

Table 114. COBOL host variable equivalents that you can use when retrieving data of a particular SQL data type

SQL data type	COBOL host variable equivalent	Notes

SMALLINT	`S9(4) COMP-4,`
	`S9(4) COMP-5,`
	`S9(4) COMP,`
	`or` `S9(4) BINARY`	

INTEGER	`S9(9) COMP-4,`
	`S9(9) COMP-5,`
	`S9(9) COMP,`
	`or` `S9(9) BINARY`	

DECIMAL(*p,s*) **or** NUMERIC(*p,s*)	`S9(p-s)V9(s) COMP-3` **`or`**
	`S9(p-s)V9(s)`
	`PACKED-DECIMAL`
	`DISPLAY SIGN`
	`LEADING SEPARATE`
	`NATIONAL SIGN`
	`LEADING SEPARATE`
		p is precision; *s* is scale. 0≤*s*≤*p*≤31. If *s*=0, use S9(*p*)V or S9(*p*). If s=p, use SV9(*s*). If the COBOL compiler does not support 31–digit decimal numbers, no exact equivalent exists. Use COMP-2.

REAL **or** FLOAT (*n*)	`COMP-1`	1≤*n*≤21
DOUBLE PRECISION, DOUBLE **or** FLOAT (*n*)	`COMP-2`	22≤*n*≤53
BIGINT	S9(18) COMP-4, S9(18) COMP-5, S9(18) COMP, or S9(18) BINARY	
CHAR(*n*)	Fixed-length character string. For example, `01 VAR-NAME PIC X(n).`	1≤*n*≤255

Table 114. COBOL host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	COBOL host variable equivalent	Notes

VARCHAR(*n*)	Varying-length character string. For example,
	`01 VAR-NAME.`
	`   49 VAR-LEN PIC S9(4)`
	` USAGE BINARY.`
	`   49 VAR-TEXT PIC X(n).`
		The inner variables must have a level of 49.

GRAPHIC(*n*)	Fixed-length graphic string. For example,
	`01 VAR-NAME PIC G(n)`
	`   USAGE IS DISPLAY-1.`
		n refers to the number of double-byte characters, not to the number of bytes. 1≤*n*≤127

VARGRAPHIC(*n*)	Varying-length graphic string. For example,
	`01 VAR-NAME.`
	`   49 VAR-LEN PIC S9(4)`
	` USAGE BINARY.`
	`   49 VAR-TEXT PIC G(n)`
	`      USAGE IS DISPLAY-1.`
		n refers to the number of double-byte characters, not to the number of bytes.
		The inner variables must have a level of 49.

BINARY(*n*)	SQL TYPE IS BINARY(*n*)	1≤*n*≤255
VARBINARY(*n*)	SQL TYPE IS VARBINARY(*n*)	1≤*n*≤32704
DATE	Fixed-length character string of length *n*. For example, `01 VAR-NAME PIC X(n).`	If you are using a date exit routine, *i* is determined by that routine. Otherwise, *i* must be at least 10.
TIME	Fixed-length character string of length *n*. For example, `01 VAR-NAME PIC X(n).`	If you are using a time exit routine, *n* is determined by that routine. Otherwise, *n* must be at least 6; to include seconds, *n* must be at least 8.
TIMESTAMP	Fixed-length character string of length *n*. For example, `01 VAR-NAME PIC X(n).`	*n* must be at least 19. To include microseconds, *n* must be 26; if *n* is less than 26, truncation occurs on the microseconds part.
TIMESTAMP(0)	Fixed-length character string of length *n*. For example, `01 VAR-NAME PIC X(n).`	*n* must be at least 19.
TIMESTAMP(p) p > 0	Fixed-length character string of length *n*. For example, `01 VAR-NAME PIC X(n`).	*n* must be at least 19. To include fractional seconds, *n* must be 20+*x* where *x* is the number of fractional seconds to include; if x is less than *p*, truncation occurs on the fractional seconds part.

Table 114. COBOL host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	COBOL host variable equivalent	Notes

TIMESTAMP(0) WITH TIME ZONE	Varying-length character string. For example,
	`01 VAR-NAME.`
	`49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n).`
		The inner variables must have a level of 49. *n* must be at least 25.

TIMESTAMP(*p*) WITH TIME ZONE	Varying-length character string. For example,
	`01 VAR-NAME.`
	`49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n).`
		The inner variables must have a level of 49. *n* must be at least 26+*p*.

Result set locator	`SQL TYPE IS RESULT-SET-LOCATOR`	Use this data type only for receiving result sets. Do not use this data type as a column type.
Table locator	`SQL TYPE IS TABLE LIKE table-name AS LOCATOR`	Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. Do not use this data type as a column type.
BLOB locator	`USAGE IS SQL TYPE IS BLOB-LOCATOR`	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.
CLOB locator	`USAGE IS SQL TYPE IS CLOB-LOCATOR`	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB locator	`USAGE IS SQL TYPE IS DBCLOB-LOCATOR`	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.
BLOB(*i*)	`USAGE IS SQL TYPE IS BLOB(i)`	1≤n≤2147483647
CLOB(*i*)	`USAGE IS SQL TYPE IS CLOB(i)`	1≤n≤2147483647
DBCLOB(*i*)	`USAGE IS SQL TYPE IS DBCLOB(i)`	*i* is the number of double-byte characters. 1≤n≤1073741823
XML	SQL TYPE IS XML AS BLOB(*i*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS CLOB(*i*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS DBCLOB(*i*)	*i* is the number of double-byte characters. 1≤n≤1073741823
BLOB file reference	`USAGE IS SQL TYPE IS BLOB-FILE`	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.

Table 114. COBOL host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	COBOL host variable equivalent	Notes
CLOB file reference	`USAGE IS SQL TYPE IS CLOB-FILE`	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.
DBCLOB file reference	`USAGE IS SQL TYPE IS DBCLOB-FILE`	Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.
XML BLOB file reference	SQL TYPE IS XML AS BLOB-FILE	Use this data type only to manipulate XML data as BLOB files. Do not use this data type as a column type.
XML CLOB file reference	SQL TYPE IS XML AS CLOB-FILE	Use this data type only to manipulate XML data as CLOB files. Do not use this data type as a column type.
XML DBCLOB file reference	SQL TYPE IS XML AS DBCLOB-FILE	Use this data type only to manipulate XML data as DBCLOB files. Do not use this data type as a column type.
ROWID	`SQL TYPE IS ROWID`	

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

Host variable data types for XML data in embedded SQL applications (Db2 Programming for XML)

Object-oriented extensions in COBOL

When you use object-oriented extensions in a COBOL application, you need to consider where to place SQL statements, the SQLCA, the SQLDA, and host variable declarations. You also need to consider the rules for host variables.

Where to place SQL statements in your application: A COBOL source data set or member can contain the following elements:

- Multiple programs
- Multiple class definitions, each of which contains multiple methods

You can put SQL statements in only the first program or class in the source data set or member. However, you can put SQL statements in multiple methods within a class. If an application consists of multiple data sets or members, each of the data sets or members can contain SQL statements.

Where to place the SQLCA, SQLDA, and host variable declarations: You can put the SQLCA, SQLDA, and SQL host variable declarations in the WORKING-STORAGE SECTION of a program, class, or method. An SQLCA or SQLDA in a class WORKING-STORAGE SECTION is global for all the methods of the class. An SQLCA or SQLDA in a method WORKING-STORAGE SECTION is local to that method only.

If a class and a method within the class both contain an SQLCA or SQLDA, the method uses the SQLCA or SQLDA that is local.

Rules for host variables: You can declare COBOL variables that are used as host variables in the WORKING-STORAGE SECTION or LINKAGE-SECTION of a program, class, or method. You can also declare host variables in the LOCAL-STORAGE SECTION of a method. The scope of a host variable is the method, class, or program within which it is defined.

Handling SQL error codes in Cobol applications

Cobol applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

To request more information about SQL errors from Cobol programs, use the following approaches:

- You can use the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement to convert an SQL return code into a text message. Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR.
- You can use the subroutine DSNTIAR to convert an SQL return code into a text message. DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program.

  DSNTIAR syntax
  DSNTIAR has the following syntax:

  ```
  CALL 'DSNTIAR' USING sqlca message lrecl.
  ```

  DSNTIAR parameters
  The DSNTIAR parameters have the following meanings:

  sqlca
  An SQL communication area.

  message
  An output area, in VARCHAR format, in which DSNTIAR places the message text. The first halfword contains the length of the remaining area; its minimum value is 240.

  The output lines of text, each line being the length specified in *lrecl*, are put into this area. For example, you could specify the format of the output area as:

  ```
  01  ERROR-MESSAGE.
          02  ERROR-LEN   PIC S9(4)  COMP VALUE +1320.
          02  ERROR-TEXT  PIC X(132) OCCURS 10 TIMES
                                     INDEXED BY ERROR-INDEX.
  77  ERROR-TEXT-LEN      PIC S9(9)  COMP VALUE +132.
  ⋮
  CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
  ```

  where ERROR-MESSAGE is the name of the message output area containing 10 lines of length 132 each, and ERROR-TEXT-LEN is the length of each line.

  lrecl
  A fullword containing the logical record length of output messages, between 72 and 240.

  An example of calling DSNTIAR from an application appears in the Db2 sample assembler program DSN8BC3, which is contained in the library DSN8B10.
- If your CICS application requires CICS storage handling, you must use the subroutine DSNTIAC instead of DSNTIAR.

  If you call DSNTIAR dynamically from a CICS COBOL application program, be sure you do the following:

  – Compile the COBOL application with the NODYNAM option.

  – Define DSNTIAR in the CSD.

DSNTIAC syntax

If your CICS application requires CICS storage handling, you must use the subroutine DSNTIAC instead of DSNTIAR. DSNTIAC has the following syntax:

```
CALL 'DSNTIAC' USING eib commarea sqlca msg lrecl.
```

DSNTIAC parameters

DSNTIAC has extra parameters, which you must use for calls to routines that use CICS commands.

eib
EXEC interface block

commarea
communication area

For more information on these parameters, see the appropriate application programming guide for CICS. The remaining parameter descriptions are the same as those for DSNTIAR. Both DSNTIAC and DSNTIAR format the SQLCA in the same way.

You must define DSNTIA1 in the CSD. If you load DSNTIAR or DSNTIAC, you must also define them in the CSD. For an example of CSD entry generation statements for use with DSNTIAC, see job DSNTEJ5A.

The assembler source code for DSNTIAC and job DSNTEJ5A, which assembles and link-edits DSNTIAC, are in the data set *prefix*.SDSNSAMP.

Related tasks

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Fortran applications that issue SQL statements

You can code SQL statements in a Fortran program wherever you can place executable statements. If the SQL statement is within an IF statement, the precompiler generates any necessary THEN and END IF statements.

Fortran source statements must be fixed-length 80-byte records. The Db2 precompiler does not support free-form source input.

Each SQL statement in a Fortran program must begin with EXEC SQL. The EXEC and SQL keywords must appear on one line, but the remainder of the statement can appear on subsequent lines.

You might code the UPDATE statement in a Fortran program as follows:

```
 EXEC SQL
C   UPDATE DSN8B10.DEPT
C   SET MGRNO = :MGRNUM
C   WHERE DEPTNO = :INTDEPT
```

You cannot follow an SQL statement with another SQL statement or Fortran statement on the same line.

Fortran does not require blanks to delimit words within a statement, but the SQL language requires blanks. The rules for embedded SQL follow the rules for SQL syntax, which require you to use one or more blanks as a delimiter.

Comments
You can include Fortran comment lines within embedded SQL statements wherever you can use a blank, except between the keywords EXEC and SQL. You can include SQL comments in any embedded SQL statement. For more information, see SQL comments (Db2 SQL).

The Db2 precompiler does not support the exclamation point (!) as a comment recognition character in Fortran programs.

Continuation for SQL statements

The line continuation rules for SQL statements are the same as those for Fortran statements, except that you must specify EXEC SQL on one line. The SQL examples in this topic have Cs in the sixth column to indicate that they are continuations of EXEC SQL.

Delimiters in Fortran

Delimit an SQL statement in your Fortran program with the beginning keyword `EXEC SQL`

and an end of line or end of last continued line.

Declaring tables and views

Your Fortran program should also include the DECLARE TABLE statement to describe each table and view the program accesses.

Dynamic SQL in a Fortran program

In general, Fortran programs can easily handle dynamic SQL statements. SELECT statements can be handled if the data types and the number of returned fields are fixed. If you want to use variable-list SELECT statements, you need to use an SQLDA, as described in "Defining SQL descriptor areas (SQLDA)" on page 476.

You can use a Fortran character variable in the statements PREPARE and EXECUTE IMMEDIATE, even if it is fixed-length.

Including code

To include SQL statements or Fortran host variable declarations from a member of a partitioned data set, use the following SQL statement in the source code where you want to include the statements:

```
EXEC SQL INCLUDE member-name
```

You cannot nest SQL INCLUDE statements. You cannot use the Fortran INCLUDE compiler directive to include SQL statements or Fortran host variable declarations.

Margins

Code the SQL statements between columns 7 through 72, inclusive. If EXEC SQL starts before the specified left margin, the Db2 precompiler does not recognize the SQL statement.

Names

You can use any valid Fortran name for a host variable. Do not use external entry names that begin with 'DSN' or host variable names that begin with 'SQL'. These names are reserved for Db2.

Do not use the word DEBUG, except when defining a Fortran DEBUG packet. Do not use the words FUNCTION, IMPLICIT, PROGRAM, and SUBROUTINE to define variables.

Sequence numbers

The source statements that the Db2 precompiler generates do not include sequence numbers.

Statement labels

You can specify statement numbers for SQL statements in columns 1 to 5. However, during program preparation, a labeled SQL statement generates a Fortran CONTINUE statement with that label before it generates the code that executes the SQL statement. Therefore, a labeled SQL statement should never be the last statement in a DO loop. In addition, you should not label SQL statements (such as INCLUDE and BEGIN DECLARE SECTION) that occur before the first executable SQL statement, because an error might occur.

WHENEVER statement

The target for the GOTO clause in the SQL WHENEVER statement must be a label in the Fortran source code and must refer to a statement in the same subprogram. The WHENEVER statement only applies to SQL statements in the same subprogram.

Special Fortran considerations

The following considerations apply to programs written in Fortran:

- You cannot use the @PROCESS statement in your source code. Instead, specify the compiler options in the PARM field.
- You cannot use the SQL INCLUDE statement to include the following statements: PROGRAM, SUBROUTINE, BLOCK, FUNCTION, or IMPLICIT.

Db2 supports Version 3 Release 1 (or later) of VS Fortran with the following restrictions:

- The parallel option is not supported. Applications that contain SQL statements must not use Fortran parallelism.
- You cannot use the byte data type within embedded SQL, because byte is not a recognizable host data type.

Handling SQL error codes
Fortran applications can request more information about SQL errors from Db2. For more information, see "Handling SQL error codes in C and C++ applications" on page 617.

Related tasks

Overview of programming applications that access Db2 for z/OS data
Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Defining the SQL communications area, SQLSTATE, and SQLCODE in Fortran

Fortran programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

About this task

If you specify the SQL processing option STDSQL(YES), do not define an SQLCA. If you do, Db2 ignores your SQLCA, and your SQLCA definition causes compile-time errors. If you specify the SQL processing option STDSQL(NO), include an SQLCA explicitly.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Procedure

Choose one of the following actions:

Option	Description
To define the SQL communications area:	a. Code the SQLCA directly in the program or use the following SQL INCLUDE statement to request a standard SQLCA declaration: `EXEC SQL INCLUDE SQLCA` Db2 sets the SQLCODE and SQLSTATE values in the SQLCA after each SQL statement executes. Your application should check these values to determine whether the last SQL statement was successful.

Option	Description
To declare SQLCODE and SQLSTATE host variables:	a. Declare the SQLCODE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as INTEGER*4. This variable can also be called SQLCOD. b. Declare the SQLSTATE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as CHARACTER*5. This variable can also be called SQLCOD. **Restriction:** Do not declare an SQLSTATE variable as an element of a structure. **Requirement:** After you declare the SQLCODE and SQLSTATE variables, ensure that all SQL statements in the program are within the scope of the declaration of these variables.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas in (SQLDA) Fortran

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Call a subroutine that is written in C, PL/I, or assembler language and that uses the INCLUDE SQLDA statement to define the SQLDA. The subroutine can also include SQL statements for any dynamic SQL functions that you need.

Restrictions:

- You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.
- You cannot use the SQL INCLUDE statement for the SQLDA, because it is not supported in COBOL.

Related tasks

Defining SQL descriptor areas (SQLDA)

If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Declaring host variables and indicator variables in Fortran

You can use host variables, host-variable arrays, and host structures in SQL statements in your program to pass data between Db2 and your application.

Procedure

To declare host variables, host-variable arrays, and host structures:

1. Declare the variables according to the following rules and guidelines:
   - When you declare a character host variable, do not use an expression to define the length of the character variable. You can use a character host variable with an undefined length (for example, CHARACTER *(*)). The length of any such variable is determined when the associated SQL statement executes.
   - Host variables must be scalar variables; they cannot be elements of vectors or arrays (subscripted variables).
   - Be careful when calling subroutines that might change the attributes of a host variable. Such alteration can cause an error while the program is running.
   - If you specify the ONEPASS SQL processing option, you must explicitly declare each host variable and each host-variable array before using them in an SQL statement. If you specify the TWOPASS precompiler option, you must declare each host variable before using it in the DECLARE CURSOR statement.
   - If you specify the STDSQL(YES) SQL processing option, you must precede the host language statements that define the host variables and host-variable arrays with the BEGIN DECLARE SECTION statement and follow the host language statements with the END DECLARE SECTION statement. Otherwise, these statements are optional.
   - Ensure that any SQL statement that uses a host variable or host-variable array is within the scope of the statement that declares that variable or array.
   - If you are using the Db2 precompiler, ensure that the names of host variables and host-variable arrays are unique within the program, even if the variables and variable arrays are in different blocks, classes, procedures, functions, or subroutines. You can qualify the names with a structure name to make them unique.
2. Optional: Define any associated indicator variables, arrays, and structures.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Host variables in Fortran

In Fortran programs, you can specify numeric, character, LOB, and ROWID host variables. You can also specify result set and LOB locators.

Restrictions:

- Only some of the valid Fortran declarations are valid host variable declarations. If the declaration for a variable is not valid, any SQL statement that references the variable might result in the message UNDECLARED HOST VARIABLE.
- Fortran supports some data types with no SQL equivalent (for example, REAL*16 and COMPLEX). In most cases, you can use Fortran statements to convert between the unsupported data types and the data types that SQL allows.
- You can not use locators as column types.

The following locator data types are Fortran data types and SQL data types:

– Result set locator
– LOB locators

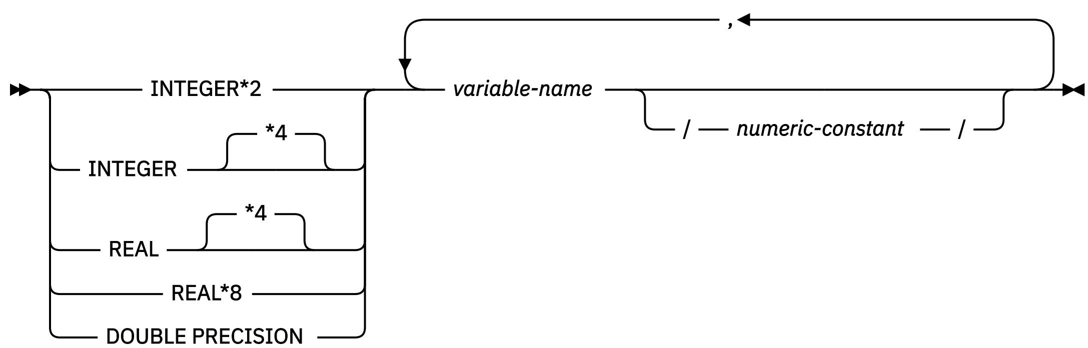
- Because Fortran does not support graphic data types, Fortran applications can process only Unicode tables that use UTF-8 encoding.

Recommendations:

- Be careful of overflow. For example, if you retrieve an INTEGER column value into a INTEGER*2 host variable and the column value is larger than 32767 or -32768, you get an overflow warning or an error, depending on whether you provided an indicator variable.
- Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a CHARACTER*70 host variable, the rightmost ten characters of the retrieved string are truncated. Retrieving a double-precision floating-point or decimal column value into an INTEGER*4 host variable removes any fractional value.

Numeric host variables

The following diagram shows the syntax for declaring numeric host variables.

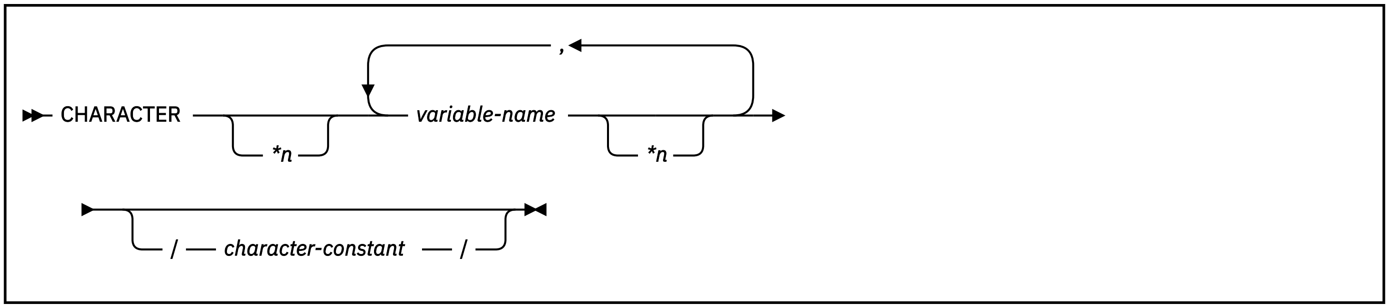


Restrictions:

- Fortran does not provide an equivalent for the decimal data type. To hold a decimal value, use one of the following variables:
  – An integer or floating-point variable, which converts the value. If you use an integer variable, you lose the fractional part of the number. If the decimal number can exceed the maximum value for an integer or you want to preserve a fractional value, use a floating-point variable. Floating-point numbers are approximations of real numbers. Therefore, when you assign a decimal number to a floating-point variable, the result might be different from the original number.
  – A character string host variable. Use the CHAR function to retrieve a decimal value into it.
- The SQL data type DECFLOAT has no equivalent in Fortran.

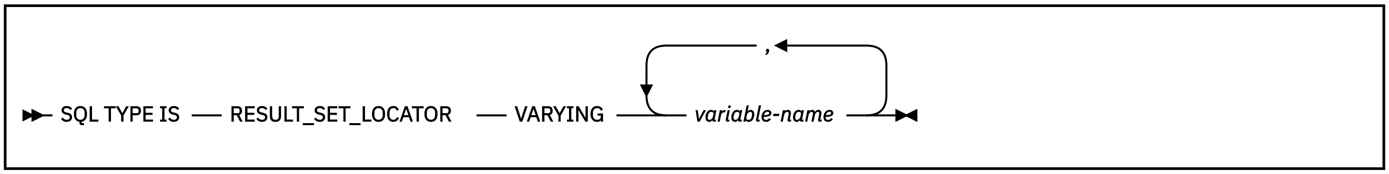
Character host variables

The following diagram shows the syntax for declaring character host variables other than CLOBs.



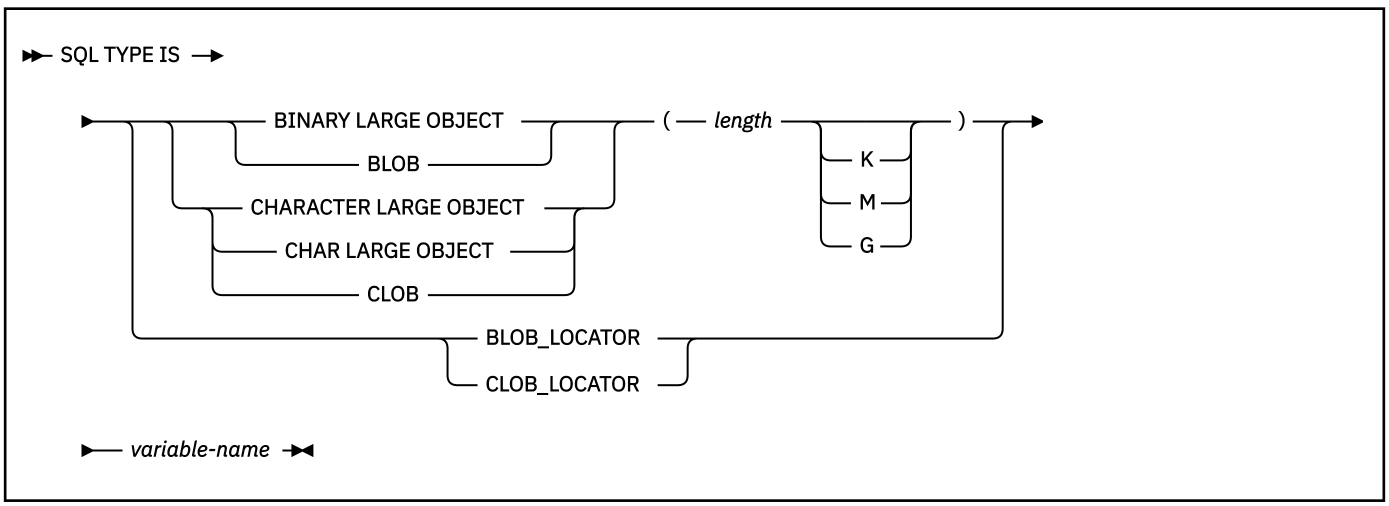
Result set locators

The following diagram shows the syntax for declaring result set locators.



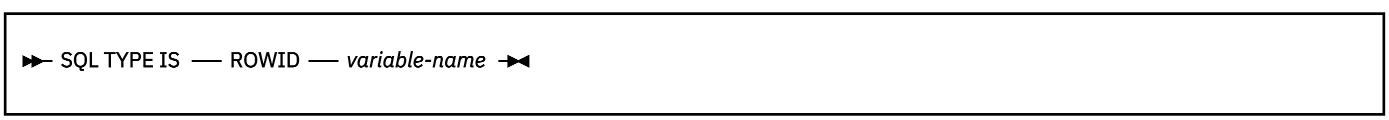
LOB variables and locators

The following diagram shows the syntax for declaring BLOB and CLOB host variables and locators.



ROWID host variables

The following diagram shows the syntax for declarations of ROWID variables.



Constants

The syntax for constants in Fortran programs differs from the syntax for constants in SQL statements in the following ways:

- Fortran interprets a string of digits with a decimal point to be a real constant. An SQL statement interprets such a string to be a decimal constant. Therefore, use exponent notation when specifying a real (that is, floating-point) constant in an SQL statement.

• In Fortran, a real (floating-point) constant that has a length of 8 bytes uses a D as the exponent indicator (for example, 3.14159D+04). An 8-byte floating-point constant in an SQL statement must use an E (for example, 3.14159E+04).

Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Related tasks

Determining whether a retrieved value in a host variable is null or truncated
Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Inserting a single row by using a host variable
Use host variables in your INSERT statement when you don't know at least some of the values to insert until the program runs.

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

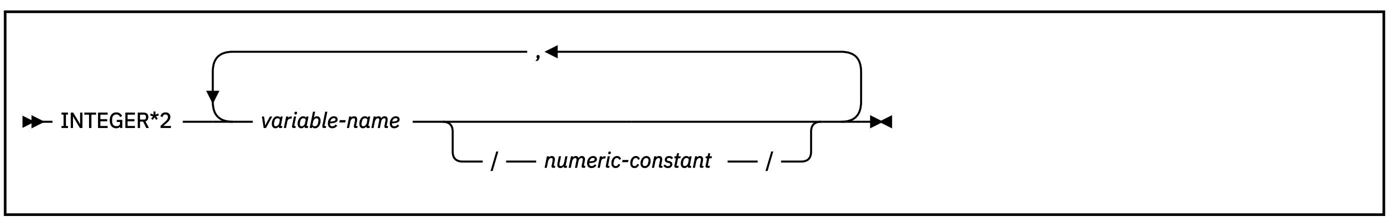
Retrieving a single row of data into host variables
If you know that your query returns only one row, you can specify one or more host variables to contain the column values of the retrieved row.

Updating data by using host variables
When you want to update a value in a Db2 table, but you do not know the exact value until the program runs, use host variables. Db2 can change a table value to match the current value of the host variable.

Indicator variables in Fortran

An indicator variable is a 2-byte integer (INTEGER*2). You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

The following diagram shows the syntax for declaring an indicator variable in Fortran.



Example

The following example shows a FETCH statement with the declarations of the host variables that are needed for the FETCH statement and their associated indicator variables.

```
 EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C                               :DAY :DAYIND,
C                               :BGN :BGNIND,
C                               :END :ENDIND
```

You can declare these variables as follows:

```
CHARACTER*7 CLSCD
INTEGER*2   DAY
CHARACTER*8 BGN, END
INTEGER*2   DAYIND, BGNIND, ENDIND
```

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related tasks

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Handling SQL error codes in Fortran applications

Fortran applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

To request more information about SQL errors from Fortran programs, use the following approaches:

- You can use the subroutine DSNTIR to convert an SQL return code into a text message. DSNTIR builds a parameter list and calls DSNTIAR for you.

  DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program. For concepts and more information on the behavior of DSNTIAR, see "Displaying SQLCA fields by calling DSNTIAR" on page 529.

  DSNTIAR syntax

  DSNTIAR has the following syntax:

  ```
  CALL DSNTIR ( error-length, message, return-code )
  ```

  DSNTIAR parameters

  The DSNTIR parameters have the following meanings:

  error-length
  The total length of the message output area.

  message
  An output area, in VARCHAR format, in which DSNTIAR places the message text. The first halfword contains the length of the remaining area; its minimum value is 240.

  The output lines of text are put into this area. For example, you could specify the format of the output area as:

  ```
  INTEGER   ERRLEN /1320/
  CHARACTER*132 ERRTXT(10)
  INTEGER   ICODE
  ⋮
  CALL DSNTIR ( ERRLEN, ERRTXT, ICODE )
  ```

  where ERRLEN is the total length of the message output area, ERRTXT is the name of the message output area, and ICODE is the return code.

  return-code
  Accepts a return code from DSNTIAR.

  An example of calling DSNTIR (which then calls DSNTIAR) from an application appears in the Db2 sample assembler program DSN8BF3, which is contained in the library DSN8B10.SDSNSAMP. See

"Sample applications supplied with Db2 for z/OS" on page 1010 for instructions on how to access and print the source code for the sample program.

- You can also use the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement to convert an SQL return code into a text message. Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR.

  For more information about GET DIAGNOSTICS, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

Related tasks

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Equivalent SQL and Fortran data types

When you declare host variables in your Fortran programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, ensure that the host variable is of an equivalent data type.

The following table describes the SQL data type and the base SQLTYPE and SQLLEN values that the precompiler uses for host variables in SQL statements.

Table 115. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in Fortran programs

Fortran host variable data type	SQLTYPE of host variable[1]	SQLLEN of host variable	SQL data type
INTEGER*2	500	2	SMALLINT
INTEGER*4	496	4	INTEGER
REAL*4	480	4	FLOAT (single precision)
REAL*8	480	8	FLOAT (double precision)
CHARACTER**n*	452	*n*	CHAR(*n*)
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Result set locator. Do not use this data type as a column type.
SQL TYPE IS BLOB_LOCATOR	960	4	BLOB locator. Do not use this data type as a column type.
SQL TYPE IS CLOB_LOCATOR	964	4	CLOB locator. Do not use this data type as a column type.
SQL TYPE IS BLOB(*n*) 1≤*n*≤2147483647	404	*n*	BLOB(*n*)
SQL TYPE IS CLOB(*n*) 1≤*n*≤2147483647	408	*n*	CLOB(*n*)
SQL TYPE IS ROWID	904	40	ROWID

Notes:

1. If a host variable includes an indicator variable, the SQLTYPE value is the base SQLTYPE value plus 1.

The following table shows equivalent Fortran host variables for each SQL data type. Use this table to determine the Fortran data type for host variables that you define to receive output from the database. For example, if you retrieve TIMESTAMP data, you can define a variable of type CHARACTER**n*.

This table shows direct conversions between SQL data types and Fortran data types. However, a number of SQL data types are compatible. When you do assignments or comparisons of data that have compatible data types, Db2 converts those compatible data types.

Table 116. Fortran host variable equivalents that you can use when retrieving data of a particular SQL data type

SQL data type	**Fortran host variable equivalent**	**Notes**
SMALLINT	INTEGER*2	
INTEGER	INTEGER*4	
BIGINT	not supported	
DECIMAL(*p,s*) **or** NUMERIC(*p,s*)	no exact equivalent	Use REAL*8
FLOAT(*n*) single precision	REAL*4	1<=*n*<=21
FLOAT(*n*) double precision	REAL*8	22<=*n*<=53
CHAR(*n*)	CHARACTER**n*	1<=*n*<=255
VARCHAR(*n*)	no exact equivalent	Use a character host variable that is large enough to contain the largest expected VARCHAR value.
BINARY	not supported	
VARBINARY	not supported	
GRAPHIC(*n*)	not supported	
VARGRAPHIC(*n*)	not supported	
DATE	CHARACTER**n*	If you are using a date exit routine, *n* is determined by that routine; otherwise, *n* must be at least 10.
TIME	CHARACTER**n*	If you are using a time exit routine, *n* is determined by that routine. Otherwise, *n* must be at least 6; to include seconds, *n* must be at least 8.
TIMESTAMP	CHARACTER**n*	*n* must be at least 19. To include microseconds, *n* must be 26; if *n* is less than 26, truncation occurs on the microseconds part.
TIMESTAMP(0)	CHARACTER**n*	*n* must be at least 19.
TIMESTAMP(*p*) *p* > 0	CHARACTER**n*	*n* must be at least 19. To include fractional seconds, *n* must be 20+*x* where *x* is the number of fractional seconds to include; if *x* is less than *p*, truncation occurs on the fractional seconds part.
TIMESTAMP(*p*) WITH TIME ZONE	no exact equivalent	Use a character host variable that is large enough to contain the largest expected timestamp with time zone value.

Table 116. Fortran host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	Fortran host variable equivalent	Notes
Result set locator	SQL TYPE IS RESULT_SET_LOCATOR	Use this data type only for receiving result sets. Do not use this data type as a column type.
BLOB locator	SQL TYPE IS BLOB_LOCATOR	Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.[1]
CLOB locator	SQL TYPE IS CLOB_LOCATOR	Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.[1]
DBCLOB locator	not supported	
BLOB(*n*)	SQL TYPE IS BLOB(*n*)	1≤n≤2147483647[1]
CLOB(*n*)	SQL TYPE IS CLOB(*n*)	1≤n≤2147483647[1]
DBCLOB(*n*)	not supported	
ROWID	SQL TYPE IS ROWID	
XML	not supported	

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

PL/I applications that issue SQL statements

You can code SQL statements in a PL/I program wherever you can use executable statements.

The first statement of the PL/I program must be the PROCEDURE statement with OPTIONS(MAIN), unless the program is a stored procedure. A stored procedure application can run as a subroutine.

Each SQL statement in a PL/I program must begin with EXEC SQL and end with a semicolon (;). The EXEC and SQL keywords must appear must appear on one line, but the remainder of the statement can appear on subsequent lines.

You might code an UPDATE statement in a PL/I program as follows:

```
EXEC SQL UPDATE DSN8B10.DEPT
           SET MGRNO = :MGR_NUM
           WHERE DEPTNO = :INT_DEPT ;
```

Comments

You can include PL/I comments in embedded SQL statements wherever you can use a blank, except between the keywords EXEC and SQL. You can also include SQL comments in any SQL statement. For more information, see SQL comments (Db2 SQL).

To include DBCS characters in comments, you must delimit the characters by a shift-out and shift-in control character; the first shift-in character in the DBCS string signals the end of the DBCS string.

Continuation for SQL statements
The line continuation rules for SQL statements are the same as those for other PL/I statements, except that you must specify EXEC SQL on one line.

Delimiters for SQL statements
Delimit an SQL statement in your PL/I program with the beginning keyword EXEC SQL and a Semicolon (;).

Declaring tables and views
Your PL/I program should include a DECLARE TABLE statement to describe each table and view the program accesses. You can use the Db2 declarations generator (DCLGEN) to generate the DECLARE TABLE statements.

Including code
You can use SQL statements or PL/I host variable declarations from a member of a partitioned data set by using the following SQL statement in the source code where you want to include the statements:

```
EXEC SQL INCLUDE member-name;
```

You cannot nest SQL INCLUDE statements. Do not use the PL/I %INCLUDE statement to include SQL statements or host variable DCL statements. You must use the PL/I preprocessor to resolve any %INCLUDE statements before you use the Db2 precompiler. Do not use PL/I preprocessor directives within SQL statements.

Margins
Code SQL statements in columns 2 through 72, unless you have specified other margins to the Db2 precompiler. If EXEC SQL starts before the specified left margin, the Db2 precompiler does not recognize the SQL statement.

Names
You can use any valid PL/I name for a host variable. Do not use external entry names or access plan names that begin with 'DSN', and do not use host variable names that begin with 'SQL'. These names are reserved for Db2.

Sequence numbers
The source statements that the Db2 precompiler generates do not include sequence numbers. IEL0378I messages from the PL/I compiler identify lines of code without sequence numbers. You can ignore these messages.

Statement labels
You can specify a statement label for executable SQL statements. However, the INCLUDE *text-file-name* and END DECLARE SECTION statements cannot have statement labels.

WHENEVER statement
The target for the GOTO clause in an SQL statement WHENEVER must be a label in the PL/I source code and must be within the scope of any SQL statements that WHENEVER affects.

Using double-byte character set (DBCS) characters
The following considerations apply to using DBCS in PL/I programs with SQL statements:

- If you use DBCS in the PL/I source, Db2 rules for the following language elements apply:
  - Graphic strings
  - Graphic string constants
  - Host identifiers
  - Mixed data in character strings
  - MIXED DATA option
- The PL/I preprocessor transforms the format of DBCS constants. If you do not want that transformation, run the Db2 precompiler **before** the preprocessor.
- If you use graphic string constants or mixed data in dynamically prepared SQL statements, and if your application requires the PL/I Version 2 (or later) compiler, the dynamically prepared statements must use the PL/I mixed constant format.

- If you prepare the statement from a host variable, change the string assignment to a PL/I mixed string.
- If you prepare the statement from a PL/I string, change that to a host variable, and then change the string assignment to a PL/I mixed string.

Example:

```
SQLSTMT = 'SELECT <dbdb> FROM table-name'M;
EXEC SQL PREPARE STMT FROM :SQLSTMT;
```

- If you want a DBCS identifier to resemble a PL/I graphic string, you must use a delimited identifier.
- If you include DBCS characters in comments, you must delimit the characters with a shift-out and shift-in control character. The first shift-in character signals the end of the DBCS string.
- You can declare host variable names that use DBCS characters in PL/I application programs. The rules for using DBCS variable names in PL/I follow existing rules for DBCS SQL ordinary identifiers, except for length. The maximum length for a host variable is 128 Unicode bytes in Db2. For information about the rules for DBCS SQL ordinary identifiers, see the information about SQL identifiers.

  Restrictions:

  - DBCS variable names must contain DBCS characters only. Mixing single-byte character set (SBCS) characters with DBCS characters in a DBCS variable name produces unpredictable results.
  - A DBCS variable name cannot continue to the next line.
- The PL/I preprocessor changes non-Kanji DBCS characters into extended binary coded decimal interchange code (EBCDIC) SBCS characters. To avoid this change, use Kanji DBCS characters for DBCS variable names, or run the PL/I compiler without the PL/I preprocessor.

Special PL/I considerations

The following considerations apply to programs written in PL/I:

- When compiling a PL/I program that includes SQL statements, you must use the PL/I compiler option CHARSET (60 EBCDIC).
- When compiling a PL/I program that uses BIGINT or LOB data types, specify the following compiler options: LIMITS(FIXEDBIN(63), FIXEDDEC(31))
- In unusual cases, the generated comments in PL/I can contain a semicolon. The semicolon generates compiler message IEL0239I, which you can ignore.
- The generated code in a PL/I declaration can contain the ADDR function of a field defined as character varying. This produces either message IBM105l l or IBM1180l W, both of which you can ignore.
- The precompiler generated code in PL/I source can contain the NULL() function. This produces message IEL0533I, which you can ignore unless you also use NULL as a PL/I variable. If you use NULL as a PL/I variable in a Db2 application, you must also declare NULL as a built-in function (DCL NULL BUILTIN;) to avoid PL/I compiler errors.
- The PL/I macro processor can generate SQL statements or host variable DCL statements if you run the macro processor before running the Db2 precompiler.

  If you use the PL/I macro processor, do not use the PL/I *PROCESS statement in the source to pass options to the PL/I compiler. You can specify the needed options on the COPTION parameter of the DSNH command or the option PARM.PLI=*options* of the EXEC statement in the DSNHPLI procedure.
- Using the PL/I multitasking facility, in which multiple tasks execute SQL statements, causes unpredictable results.
- PL/I WIDECHAR host data type is supported through the Db2 coprocessor only.

- When you use PL/I WX widechar constant, Db2 supports only bigendian format. Thus, when you assign a constant to the widechar type host variable in PL/I, ensure that bigendian format is used. For example:

  ```
  HVWC1 = '003100320033006100620063'WX;
  ```

  Equivalent to:

  ```
  HVWC1 = '123abc';
  ```

  HVWC1 is defined as a WIDECHAR type host variable.
- PL/I SQL Preprocessor option, CCSID0 and NOCCSID0, usage consideration when used with the Db2 coprocessor.
  - When you use CCSID0 (default), it promotes compatibility with older PL/I programs, which used the Db2 precompiler. During program preparation, no CCSID value is associated with the host variable except for the WIDECHAR type host variable. For WIDECHAR type host variable, CCSID 1200 is always assigned by the PL/I SQL Preprocessor.

    During BIND and runtime, if no CCSID is associated with the host variable, the BIND option, ENCODING, which is meant for the application data, is used. If the ENCODING BIND option is not specified, then the default value for the ENCODING BIND option is used.
  - When you use NOCCSID0, a CCSID is associated with the host variable during program preparation. The CCSID is derived from the following items during program preparation:
    - DECLARE :hv VARIABLE CCSID *xxxx* specified.
    - Source CCSID, if no DECLARE VARIABLE ... CCSID *xxxx* is specified for the host variable. During BIND time, note the CCSID assigned to the host variable during program preparation is not known to the BIND process. For more information about BIND time CCSID resolution, see Determining the encoding scheme and CCSID of a string (Introduction to Db2 for z/OS).

      For host variable used in static SQL, ensuring accurate and matching CCSID is assigned/derived through DECLARE VARIABLE ... CCSID *xxxx*, source CCSID or ENCODING BIND option or the installation default

      For parameter marker used in dynamic SQL, ensuring accurate CCSID for the corresponding host variable is assigned/derived through DECLARE VARIABLE ... CCSID *xxxx*, ENCODING BIND option or the installation default. The source CCSID has no influence on parameter marker.

Handling SQL error codes

PLI/I applications can request more information about SQL errors from Db2. For more information, see “Handling SQL error codes in PL/I applications” on page 702.

Related concepts

DCLGEN (declarations generator)
Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

SQL identifiers (Db2 SQL)

Related tasks

Overview of programming applications that access Db2 for z/OS data
Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program

Dynamic SQL is prepared and executed while the program is running.

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

PL/I programming examples

You can write Db2 programs in PL/I. These programs can access a local or remote Db2 subsystem and can execute static or dynamic SQL statements. This information contains several such programming examples.

To prepare and run these applications, use the JCL in *prefix*.SDSNSAMP as a model for your JCL.

Related reference
Assembler, C, C++, COBOL, PL/I, and REXX programming examples (Db2 Programming samples)

Example PL/I program that calls a stored procedure

You can call the GETPRML stored procedure that uses the GENERAL WITH NULLS linkage convention from a PL/I program on a z/OS system.

The following figure contains the example PL/I program that calls the GETPRML stored procedure.

```
*PROCESS SYSTEM(MVS);
 CALPRML:
   PROC OPTIONS(MAIN);

   /************************************************************/
   /* Declare the parameters used to call the GETPRML          */
   /* stored procedure.                                        */
   /************************************************************/
   DECLARE PROCNM CHAR(18),    /* INPUT parm -- PROCEDURE name */
           SCHEMA CHAR(8),     /* INPUT parm -- User's schema  */
           OUT_CODE FIXED BIN(31),
                               /* OUTPUT -- SQLCODE from the   */
                               /*   SELECT operation.          */
           PARMLST CHAR(254)   /* OUTPUT -- RUNOPTS for        */
                   VARYING,    /*   the matching row in the    */
                               /*   catalog table SYSROUTINES  */
           PARMIND FIXED BIN(15);
                               /* PARMLST indicator variable   */
   /************************************************************/
   /* Include the SQLCA                                        */
   /************************************************************/
   EXEC SQL INCLUDE SQLCA;
   /************************************************************/
   /* Call the GETPRML stored procedure to retrieve the        */
   /* RUNOPTS values for the stored procedure.   In this       */
   /* example, we request the RUNOPTS values for the           */
   /* stored procedure named DSN8EP2.                          */
   /************************************************************/
   PROCNM = 'DSN8EP2';
              /* Input parameter -- PROCEDURE to be found   */
   SCHEMA = ' ';
              /* Input parameter -- SCHEMA in SYSROUTINES   */
   PARMIND = -1; /* The PARMLST parameter is an output parm.   */
              /* Mark PARMLST parameter as null, so the DB2 */
              /* requester does not have to send the entire  */
              /* PARMLST variable to the server.  This      */
              /* helps reduce network I/O time, because     */
              /* PARMLST is fairly large.                   */
   EXEC SQL
     CALL GETPRML(:PROCNM,
                  :SCHEMA,
                  :OUT_CODE,
                  :PARMLST INDICATOR :PARMIND);
```

```
   IF SQLCODE¬=0 THEN         /* If SQL CALL failed,           */
     DO;
       PUT SKIP EDIT('SQL CALL failed due to SQLCODE = ',
           SQLCODE) (A(34),A(14));
       PUT SKIP EDIT('SQLERRM = ',
           SQLERRM) (A(10),A(70));
     END;
   ELSE                       /* If the CALL worked,           */
     IF OUT_CODE¬=0 THEN      /* Did GETPRML hit an error?     */
       PUT SKIP EDIT('GETPRML failed due to RC = ',
           OUT_CODE) (A(33),A(14));
     ELSE                     /* Everything worked.            */
       PUT SKIP EDIT('RUNOPTS = ', PARMLST) (A(11),A(200));
   RETURN;
 END CALPRML;
```

Figure 40. Calling a stored procedure from a PL/I program

Example PL/I stored procedure with a GENERAL linkage convention

You can call a stored procedure that uses the GENERAL linkage convention from a PL/I program.

This example stored procedure searches the Db2 SYSIBM.SYSROUTINES catalog table for a row that matches the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.

The linkage convention for this stored procedure is GENERAL.

The output parameters from this stored procedure contain the SQLCODE from the SELECT operation, and the value of the RUNOPTS column retrieved from the SYSIBM.SYSROUTINES table.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE PLI
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL
  STAY RESIDENT NO
  RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 0
  COMMIT ON RETURN NO;
```

The following example is a PL/I stored procedure with linkage convention GENERAL.

```
*PROCESS SYSTEM(MVS);

 GETPRML:
   PROC(PROCNM, SCHEMA, OUT_CODE, PARMLST)
                       OPTIONS(MAIN NOEXECOPS REENTRANT);

   DECLARE PROCNM CHAR(18),    /* INPUT parm -- PROCEDURE name */
           SCHEMA CHAR(8),     /* INPUT parm -- User's SCHEMA  */

           OUT_CODE FIXED BIN(31), /* OUTPUT -- SQLCODE from   */
                               /*   the SELECT operation.  */
           PARMLST CHAR(254)       /* OUTPUT -- RUNOPTS for    */
                   VARYING;        /*  the matching row in     */
                               /*  SYSIBM.SYSROUTINES      */

   EXEC SQL INCLUDE SQLCA;

   /************************************************************/
   /* Execute SELECT from SYSIBM.SYSROUTINES in the catalog.   */
   /************************************************************/
   EXEC SQL
     SELECT RUNOPTS INTO :PARMLST
         FROM SYSIBM.SYSROUTINES
         WHERE NAME=:PROCNM AND
               SCHEMA=:SCHEMA;

   OUT_CODE = SQLCODE;         /* return SQLCODE to caller     */
   RETURN;
 END GETPRML;
```

Example PL/I stored procedure with a GENERAL WITH NULLS linkage convention

You can call a stored procedure that uses the GENERAL WITH NULLS linkage convention from a PL/I program.

This example stored procedure searches the Db2 SYSIBM.SYSROUTINES catalog table for a row that matches the input parameters from the client program. The two input parameters contain values for NAME and SCHEMA.

The linkage convention for this stored procedure is GENERAL WITH NULLS.

The output parameters from this stored procedure contain the SQLCODE from the SELECT operation, and the value of the RUNOPTS column retrieved from the SYSIBM.SYSROUTINES table.

The CREATE PROCEDURE statement for this stored procedure might look like this:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE PLI
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME "GETPRML"
  COLLID GETPRML
  ASUTIME NO LIMIT
```

```
   PARAMETER STYLE GENERAL WITH NULLS
   STAY RESIDENT NO
   RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
   WLM ENVIRONMENT SAMPPROG
   PROGRAM TYPE MAIN
   SECURITY DB2
   RESULT SETS 0
   COMMIT ON RETURN NO;
```

The following example is a PL/I stored procedure with linkage convention GENERAL WITH NULLS.

```
*PROCESS SYSTEM(MVS);

 GETPRML:
   PROC(PROCNM, SCHEMA, OUT_CODE, PARMLST, INDICATORS)
                       OPTIONS(MAIN NOEXECOPS REENTRANT);

   DECLARE PROCNM CHAR(18),    /* INPUT parm -- PROCEDURE name */
           SCHEMA CHAR(8),     /* INPUT parm -- User's schema  */

           OUT_CODE FIXED BIN(31), /* OUTPUT -- SQLCODE from   */
                               /*   the SELECT operation.  */
           PARMLST CHAR(254)       /* OUTPUT -- PARMLIST for   */
                   VARYING;        /*  the matching row in     */
                                   /*  SYSIBM.SYSROUTINES      */
   DECLARE 1 INDICATORS,       /* Declare null indicators for  */
                               /* input and output parameters. */
            3 PROCNM_IND   FIXED BIN(15),
            3 SCHEMA_IND   FIXED BIN(15),
            3 OUT_CODE_IND FIXED BIN(15),
            3 PARMLST_IND  FIXED BIN(15);

   EXEC SQL INCLUDE SQLCA;

   IF PROCNM_IND<0 |
      SCHEMA_IND<0 THEN
    DO;                      /* If any input parm is NULL,     */
      OUT_CODE = 9999;       /* Set output return code.        */
      OUT_CODE_IND = 0;
                             /* Output return code is not NULL.*/
      PARMLST_IND = -1;      /* Assign NULL value to PARMLST.  */
    END;
   ELSE                      /* If input parms are not NULL,   */
    DO;                      /*                                */
   /************************************************************/
   /* Issue the SQL SELECT against the SYSIBM.SYSROUTINES      */
   /* DB2 catalog table.                                       */
   /************************************************************/
       EXEC SQL
         SELECT RUNOPTS INTO :PARMLST
             FROM SYSIBM.SYSROUTINES
             WHERE NAME=:PROCNM AND
                   SCHEMA=:SCHEMA;
       PARMLST_IND = 0;      /* Mark PARMLST as not NULL.      */

       OUT_CODE = SQLCODE;   /* return SQLCODE to caller       */
       OUT_CODE_IND = 0;
       OUT_CODE_IND = 0;     /* Output return code is not NULL.*/
    END;
   RETURN;

 END GETPRML;
```

Handling SQL error codes in PL/I applications

PL/I applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

To request information about SQL errors in PL/I programs, use the following approaches:

- You can use the subroutine DSNTIAR to convert an SQL return code into a text message.

DSNTIAR takes data from the SQLCA, formats it into a message, and places the result in a message output area that you provide in your application program. For concepts and more information on the behavior of DSNTIAR, see "Displaying SQLCA fields by calling DSNTIAR" on page 529.

DSNTIAR syntax

```
CALL DSNTIAR ( sqlca, message, lrecl );
```

DSNTIAR parameters

The DSNTIAR parameters have the following meanings:

sqlca

An SQL communication area.

message

An output area, in VARCHAR format, in which DSNTIAR places the message text. The first halfword contains the length of the remaining area; its minimum value is 240.

The output lines of text, each line being the length specified in *lrecl*, are put into this area. For example, you could specify the format of the output area as:

```
DCL DATA_LEN FIXED BIN(31) INIT(132);
DCL DATA_DIM FIXED BIN(31) INIT(10);
DCL 1 ERROR_MESSAGE AUTOMATIC,
      3 ERROR_LEN    FIXED BIN(15) UNAL INIT((DATA_LEN*DATA_DIM)),
      3 ERROR_TEXT(DATA_DIM) CHAR(DATA_LEN);
⋮
CALL DSNTIAR ( SQLCA, ERROR_MESSAGE, DATA_LEN );
```

where ERROR_MESSAGE is the name of the message output area, DATA_DIM is the number of lines in the message output area, and DATA_LEN is the length of each line.

lrecl

A fullword containing the logical record length of output messages, between 72 and 240.

Because DSNTIAR is an assembler language program, you must include the following directives in your PL/I application:

```
DCL DSNTIAR ENTRY OPTIONS (ASM,INTER,RETCODE);
```

An example of calling DSNTIAR from an application appears in the Db2 sample assembler program DSN8BP3, contained in the library DSN8B10.SDSNSAMP. See "Sample applications supplied with Db2 for z/OS" on page 1010 for instructions on how to access and print the source code for the sample program.

- If your CICS application requires CICS storage handling, you must use the subroutine DSNTIAC instead of DSNTIAR.

DSNTIAC syntax

DSNTIAC has the following syntax:

```
CALL DSNTIAC (eib, commarea, sqlca, msg, lrecl);
```

DSNTIAC parameters

DSNTIAC has extra parameters, which you must use for calls to routines that use CICS commands.

eib

EXEC interface block

commarea

communication area

For more information on these parameters, see the appropriate application programming guide for CICS. The remaining parameter descriptions are the same as those for DSNTIAR. Both DSNTIAC and DSNTIAR format the SQLCA in the same way.

You must define DSNTIA1 in the CSD. If you load DSNTIAR or DSNTIAC, you must also define them in the CSD. For an example of CSD entry generation statements for use with DSNTIAC, see job DSNTEJ5A.

The assembler source code for DSNTIAC and job DSNTEJ5A, which assembles and link-edits DSNTIAC, are in the data set *prefix*.SDSNSAMP.

- You can also use the MESSAGE_TEXT condition item field of the GET DIAGNOSTICS statement to convert an SQL return code into a text message. Programs that require long token message support should code the GET DIAGNOSTICS statement instead of DSNTIAR.

  For more information about GET DIAGNOSTICS, see "Checking the execution of SQL statements by using the GET DIAGNOSTICS statement " on page 534.

Related tasks

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I

PL/I programs that contain SQL statements can include an SQL communications area (SQLCA) to check whether an SQL statement executed successfully. Alternatively, these programs can declare individual SQLCODE and SQLSTATE host variables.

About this task

If you specify the SQL processing option STDSQL(YES), do not define an SQLCA. If you do, Db2 ignores your SQLCA, and your SQLCA definition causes compile-time errors. If you specify the SQL processing option STDSQL(NO), include an SQLCA explicitly.

If your application contains SQL statements and does not include an SQL communications area (SQLCA), you must declare individual SQLCODE and SQLSTATE host variables. Your program can use these variables to check whether an SQL statement executed successfully.

Procedure

Choose one of the following actions:

Option	Description
To define the SQL communications area:	a. Code the SQLCA directly in the program or use the following SQL INCLUDE statement to request a standard SQLCA declaration: `EXEC SQL INCLUDE SQLCA` Db2 sets the SQLCODE and SQLSTATE values in the SQLCA after each SQL statement executes. Your application should check these values to determine whether the last SQL statement was successful.
To declare SQLCODE and SQLSTATE host variables:	a. Declare the SQLCODE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as BIN FIXED (31). b. Declare the SQLSTATE variable within a BEGIN DECLARE SECTION statement and an END DECLARE SECTION statement in your program declarations as CHARACTER(5). **Restriction:** Do not declare an SQLSTATE variable as an element of a structure.

Option	Description
	Requirement: After you declare the SQLCODE and SQLSTATE variables, ensure that all SQL statements in the program are within the scope of the declaration of these variables.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas (SQLDA) in PL/I

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Code the SQLDA directly in the program, or use the following SQL INCLUDE statement to request a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA
```

Restriction: You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Declaring host variables and indicator variables in PL/I

You can use host variables, host-variable arrays, and host structures in SQL statements in your program to pass data between Db2 and your application.

Procedure

To declare host variables, host-variable arrays, and host structures:

1. Declare the variables according to the following rules and guidelines:

- If you specify the ONEPASS SQL processing option, you must explicitly declare each host variable and each host-variable array before using them in an SQL statement. If you specify the TWOPASS precompiler option, you must declare each host variable before using it in the DECLARE CURSOR statement.
- If you specify the STDSQL(YES) SQL processing option, you must precede the host language statements that define the host variables and host-variable arrays with the BEGIN DECLARE SECTION statement and follow the host language statements with the END DECLARE SECTION statement. Otherwise, these statements are optional.
- Ensure that any SQL statement that uses a host variable or host-variable array is within the scope of the statement that declares that variable or array.
- If you are using the Db2 precompiler, ensure that the names of host variables and host-variable arrays are unique within the program, even if the variables and variable arrays are in different blocks, classes, procedures, functions, or subroutines. You can qualify the names with a structure name to make them unique.

2. Optional: Define any associated indicator variables, arrays, and structures.

Related tasks

Declaring host variables and indicator variables
You can use host variables and indicator variables in SQL statements in your program to pass data between Db2 and your application.

Host variables in PL/I

In PL/I programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host variables. You can also specify result set, table, and LOB locators and LOB and XML file reference variables.

Restrictions:

- Only some of the valid PL/I declarations are valid host variable declarations. The precompiler uses the data attribute defaults that are specified in the PL/I DEFAULT statement. If the declaration for a host variable is not valid, any SQL statement that references the variable might result in the message UNDECLARED HOST VARIABLE.
- The alignment, scope, and storage attributes of host variables have the following restrictions:
  - A declaration with the EXTERNAL scope attribute and the STATIC storage attribute must also have the INITIAL storage attribute.
  - If you use the BASED storage attribute, you must follow it with a PL/I element-locator-expression.
  - Host variables can be STATIC, CONTROLLED, BASED, or AUTOMATIC storage class, or options. However, CICS requires that programs be reentrant.

  Although the precompiler uses only the names and data attributes of variables and ignores the alignment, scope, and storage attributes, you should not ignore these restrictions. If you do ignore them, you might have problems compiling the PL/I source code that the precompiler generates.
- PL/I supports some data types with no SQL equivalent (COMPLEX and BIT variables, for example). In most cases, you can use PL/I statements to convert between the unsupported PL/I data types and the data types that SQL supports.
- You can not use locators as column types.

  The following locator data types are PL/I data types as well as SQL data types:
  - Result set locator
  - Table locator
  - LOB locators
- The precompiler does not support PL/I scoping rules.

Recommendations:

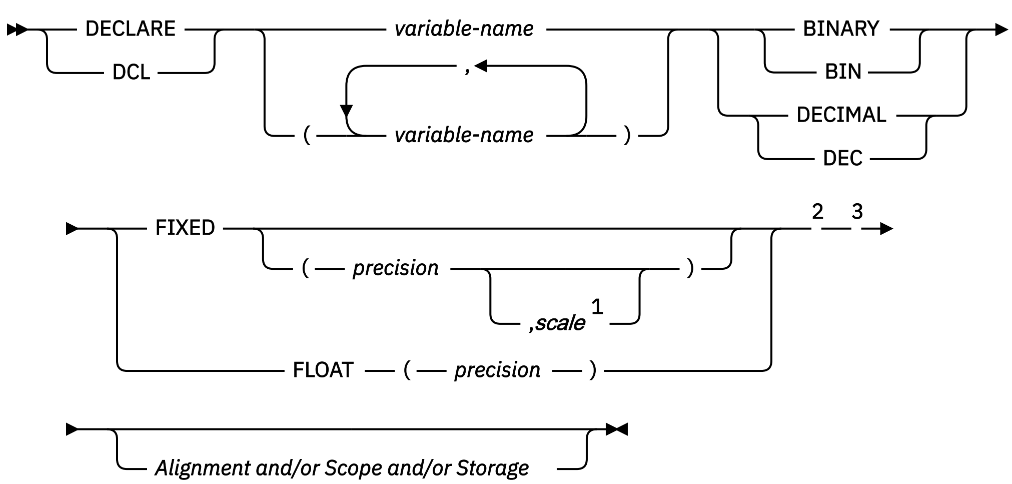
- Be careful of overflow. For example, if you retrieve an INTEGER column value into a BIN FIXED(15) host variable and the column value is larger than 32767 or smaller than -32768, you get an overflow warning or an error, depending on whether you provided an indicator variable.
- Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a CHAR(70) host variable, the rightmost ten characters of the retrieved string are truncated. Retrieving a double-precision floating-point or decimal column value into a BIN FIXED(31) host variable removes any fractional part of the value. Similarly, retrieving a column value with a DECIMAL data type into a PL/I decimal variable with a lower precision might truncate the value.

Numeric host variables

You can specify the following forms of numeric host variables:

- Floating-point numbers (Hexadecimal and Decimal)
- Integers and small integers
- Decimal numbers

The following diagram shows the syntax for declaring numeric host variables.



Notes:

[1] You can specify a *scale* only for DECIMAL FIXED.

[2] You can specify host variable attributes in any order that is acceptable to PL/I. For example, BIN FIXED(31), BINARY FIXED(31), BIN(31) FIXED, and FIXED BIN(31) are all acceptable.

[3] The UNSIGNED attribute must not be specified with BINARY and FIXED for a numeric host variable declaration.

For binary floating-point or hexadecimal floating-point data types, use the FLOAT SQL processing option to specify whether the host variable is in IEEE binary floating-point or z/Architecture hexadecimal floating-point format. Db2 does not check if the format of the host variable contents match the format that you specified with the FLOAT SQL processing option. Therefore, you need to ensure that your floating-point host variable contents match the format that you specified with the FLOAT SQL processing option. Db2 converts all floating-point input data to z/Architecture hexadecimal floating-point format before storing it.

If the PL/I compiler that you are using does not support a decimal data type with a precision greater than 15, use one of the following variable types for decimal data:

- Decimal variables with precision less than or equal to 15, if the actual data values fit. If you retrieve a decimal value into a decimal variable with a scale that is less than the source column in the database, the fractional part of the value might truncate.

- An integer or a floating-point variable, which converts the value. If you use an integer variable, you lose the fractional part of the number. If the decimal number can exceed the maximum value for an integer or you want to preserve a fractional value, use a floating-point variable. Floating-point numbers are approximations of real numbers. Therefore, when you assign a decimal number to a floating-point variable, the result might be different from the original number.
- A character string host variable. Use the CHAR function to retrieve a decimal value into it.

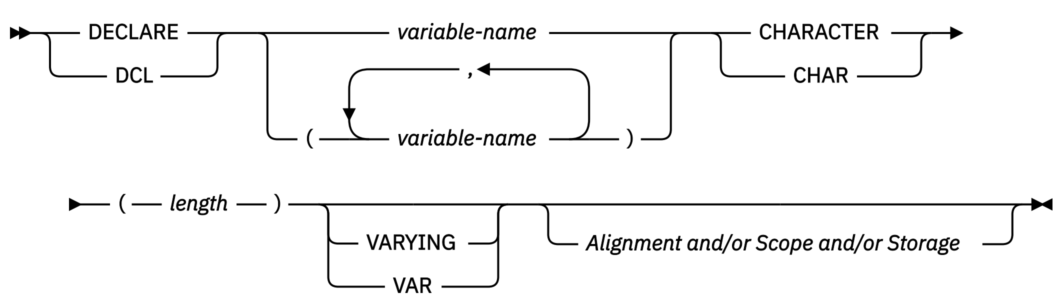
To use the PL/I decimal floating-point host data types, you need to use the FLOAT(DFP) and ARCH(7) compiler options and the Db2 coprocessor. The maximum precision for extended DECIMAL FLOAT will be 34 (not 33 as it is for hexadecimal float). The maximum precision for short DECIMAL FLOAT will be 7 (not 6 as it is for hexadecimal float).

Character host variables

You can specify the following forms of character host variables:

- Fixed-length strings
- Varying-length strings
- CLOBs

The following diagram shows the syntax for declaring character host variables, other than CLOBs.

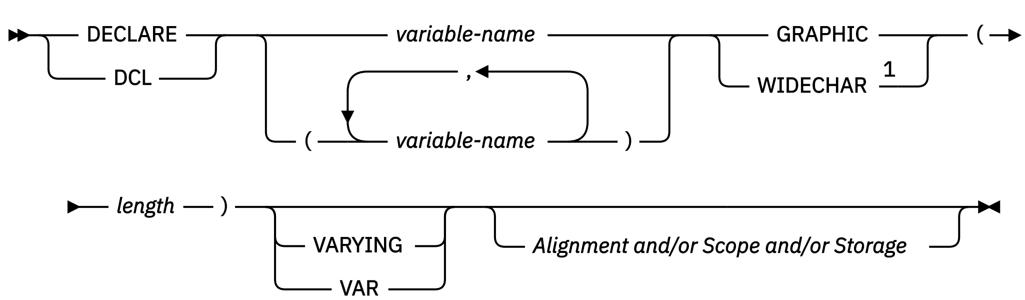


Graphic host variables

You can specify the following forms of character host variables:

- Fixed-length strings
- Varying-length strings
- DBCLOBs

The following diagram shows the syntax for declaring graphic host variables other than DBCLOBs.



Notes:

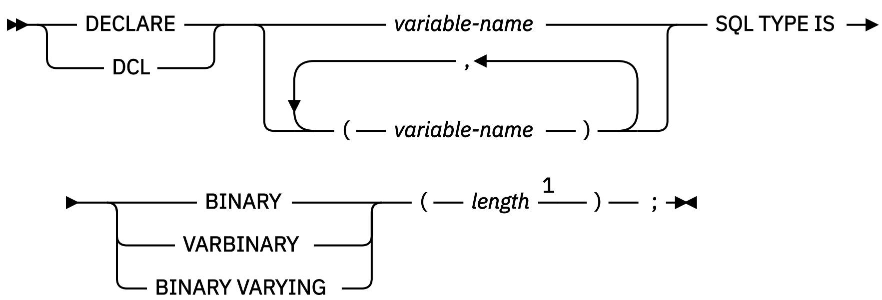
[1] Use WIDECHAR only for UNICODE UTF-16 data. WIDECHAR is supported only by the Db2 coprocessor.

Binary host variables

You can specify the following forms of binary host variables:

- Fixed-length strings
- Varying-length strings
- BLOBs

The following diagram shows the syntax for declaring BINARY host variables.



Notes:

[1] For BINARY host variables, the length must be in the range 1 - 255. For VARBINARY host variables, the length must be in the range 1 - 32704.

PL/I does not have variables that correspond to the SQL binary data types BINARY and VARBINARY. To create host variables that can be used with these data types, use the SQL TYPE IS clause.

When you reference a BINARY or VARBINARY host variable in an SQL statement, you must use the variable that you specify in the SQL TYPE declaration. When you reference the host variable in a host language statement, you must use the variable that Db2 generates.

Examples of binary variable declarations

The following table shows examples of variables that Db2 generates when you declare binary host variables.

Table 117. Examples of BINARY and VARBINARY variable declarations for PL/I

Variable declaration that you include in your PL/I program	Corresponding variable that Db2 generates in the output source member
`DCL BIN_VAR SQL TYPE IS BINARY(10);`	`DCL BIN_VAR CHAR(10);`
`DCL VBIN_VAR SQL TYPE IS VARBINARY(10);`	`DCL VBIN_VAR CHAR(10) VAR;`

Result set locators

The following diagram shows the syntax for declaring result set locators.

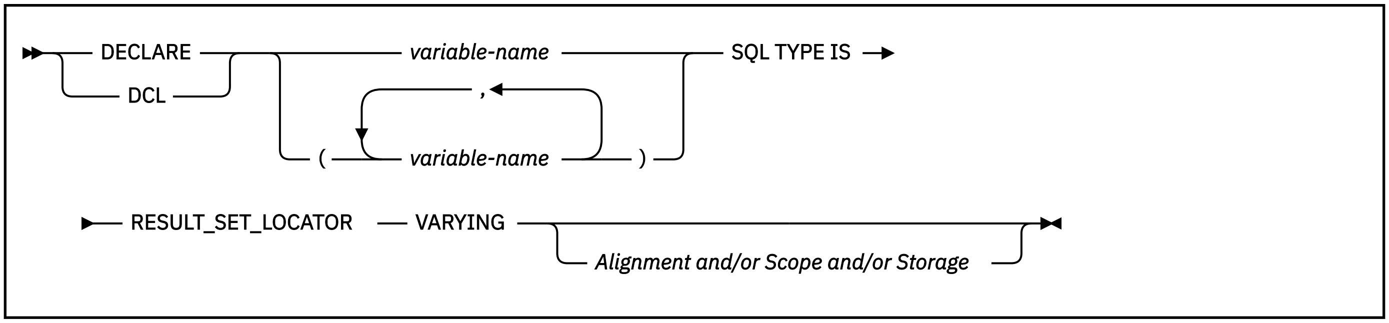
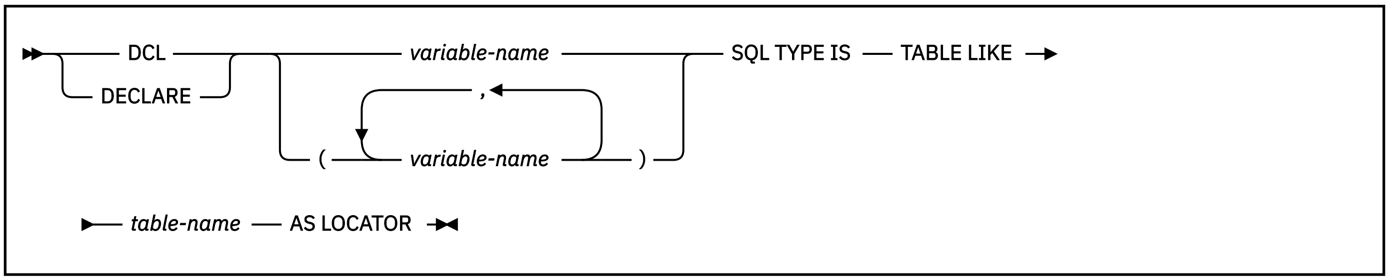


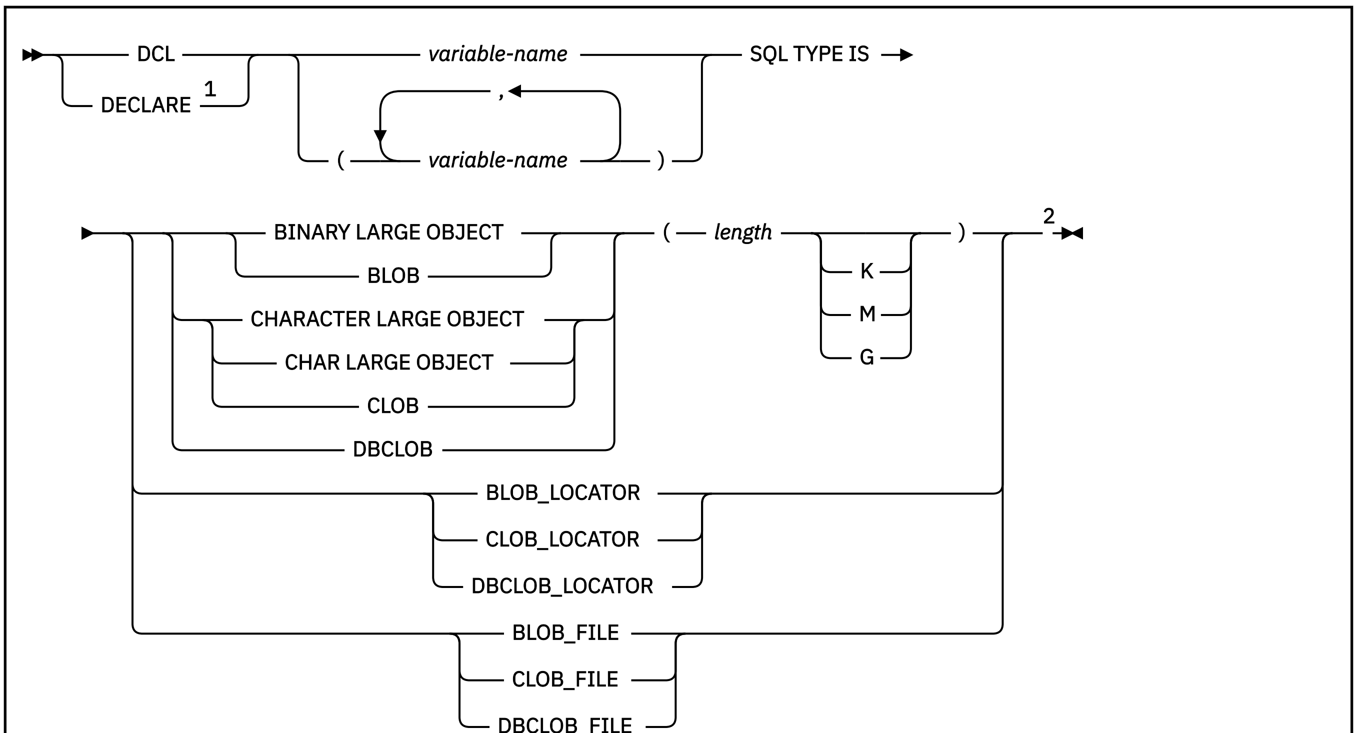
Table locators

The following diagram shows the syntax for declaring table locators.



LOB variables, locators, and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables, locators, and file reference variables.



Notes:

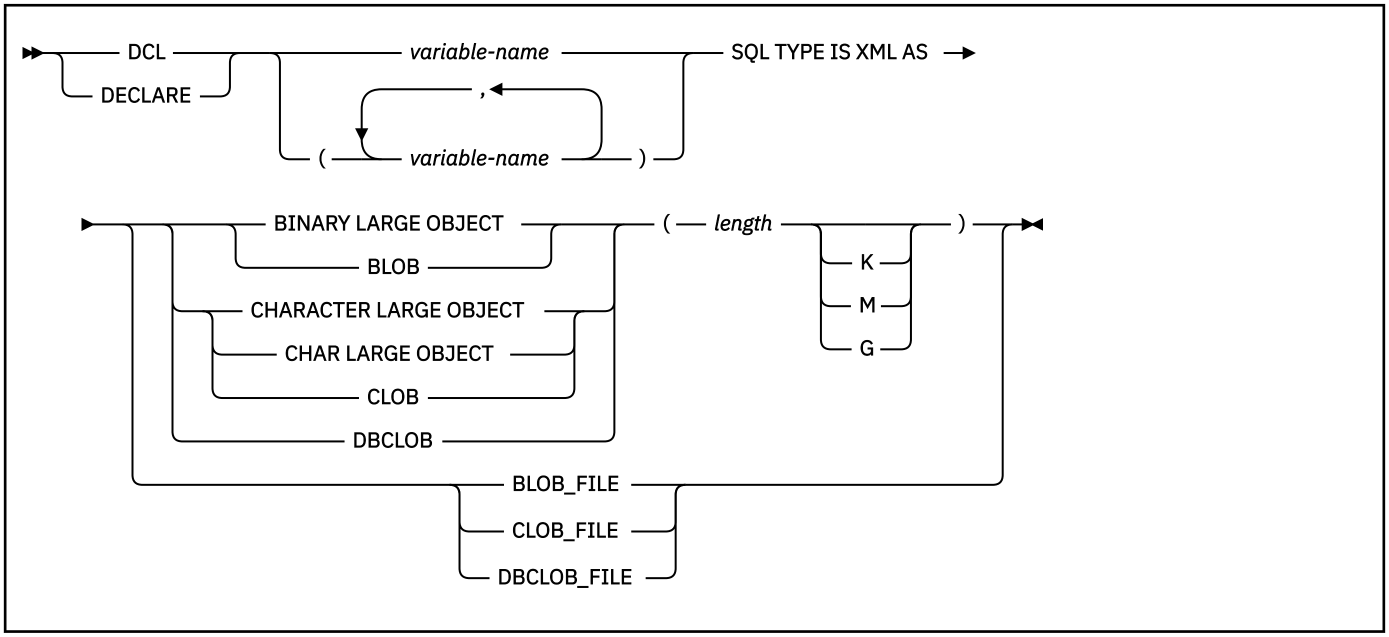
[1] A single PL/I declaration that contains a LOB variable declaration is limited to no more than 1000 lines of source code.

[2] Variable attributes such as STATIC and AUTOMATIC are ignored if specified on a LOB variable declaration.

Note: Variable attributes such as STATIC and AUTOMATIC are ignored if specified on a LOB variable declaration.

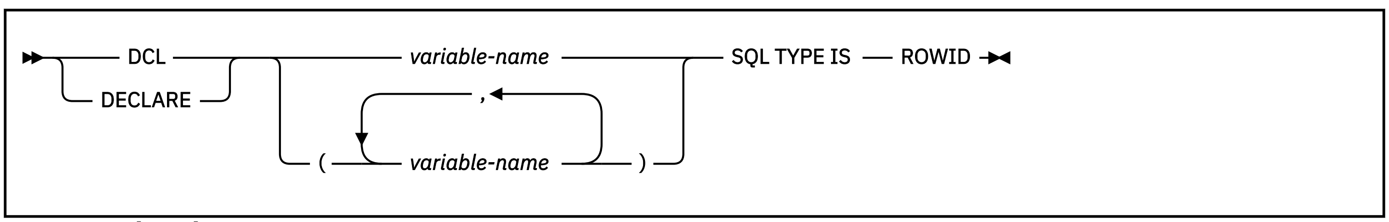
XML data host and file reference variables

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variables and file reference variables for XML data types.



ROWID host variables

The following diagram shows the syntax for declaring ROWID host variables.



Related concepts

Host variables
Use host variables to pass a single data item between Db2 and your application.

Using host variables in SQL statements
Use scalar host variables in embedded SQL statements to represent a single value. Host variables are useful for storing retrieved data or for passing values that are to be assigned or used for comparisons.

Decimal floating-point (DECFLOAT) (Db2 SQL)

Related tasks

Determining whether a retrieved value in a host variable is null or truncated
Before your application manipulates the data that was retrieved from Db2 into a host variable, determine if the value is null. Also determine if it was truncated when assigned to the variable. You can use indicator variables to obtain this information.

Inserting a single row by using a host variable
Use host variables in your INSERT statement when you don't know at least some of the values to insert until the program runs.

Inserting null values into columns by using indicator variables or arrays

If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Retrieving a single row of data into host variables
If you know that your query returns only one row, you can specify one or more host variables to contain the column values of the retrieved row.

Retrieving a single row of data into a host structure
If you know that your query returns multiple column values for only one row, you can specify a host structure to contain the column values.

Updating data by using host variables
When you want to update a value in a Db2 table, but you do not know the exact value until the program runs, use host variables. Db2 can change a table value to match the current value of the host variable.

Host-variable arrays in PL/I

In PL/I programs, you can specify numeric, character, graphic, binary, LOB, XML, and ROWID host-variable arrays. You can also specify LOB locators and LOB and XML file reference variables.

Host-variable arrays can be referenced only as a simple reference in the following contexts. In syntax diagrams, *host-variable-array* designates a reference to a host-variable array.

- In a FETCH statement for a multiple-row fetch. See FETCH (Db2 SQL).
- In an INSERT statement with multiple rows of source data. This is also referred to as multiple-row insert. See INSERT (Db2 SQL).
- In a MERGE statement with multiple rows of source data. See MERGE (Db2 SQL).
- In an EXECUTE statement to provide a value for a parameter marker in a dynamic multi-row INSERT or MERGE. See EXECUTE (Db2 SQL).

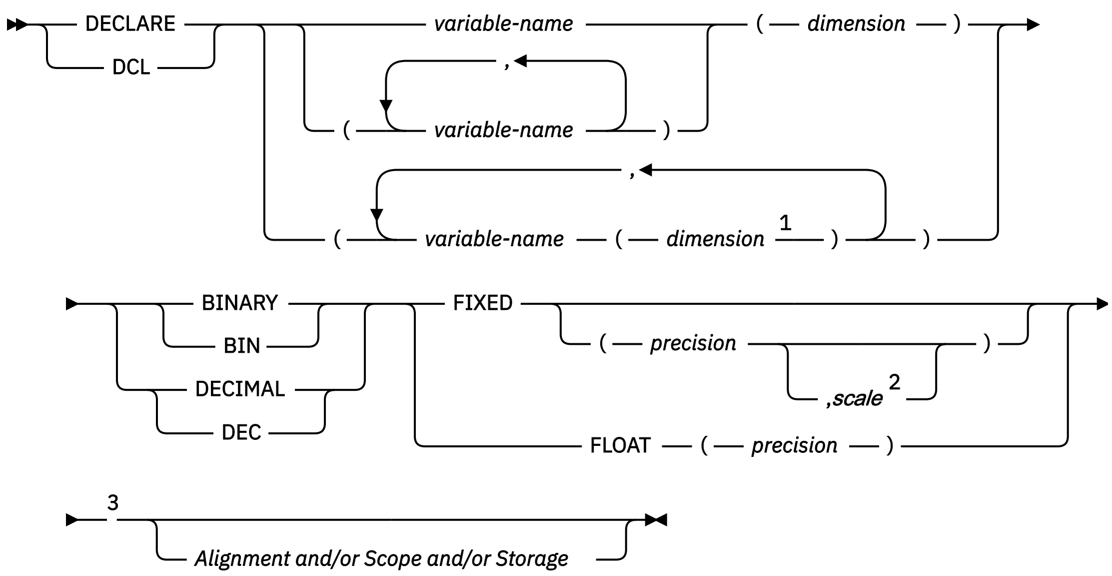
Restrictions:

- Only some of the valid PL/I declarations are valid host variable declarations. The precompiler uses the data attribute defaults that are specified in the PL/I DEFAULT statement. If the declaration for a host variable is not valid, any SQL statement that references the host-variable array might result in the message UNDECLARED HOST VARIABLE ARRAY.
- The alignment, scope, and storage attributes of host-variable arrays have the following restrictions:
  - A declaration with the EXTERNAL scope attribute and the STATIC storage attribute must also have the INITIAL storage attribute.
  - If you use the BASED storage attribute, you must follow it with a PL/I element-locator-expression.
  - Host variables can be STATIC, CONTROLLED, BASED, or AUTOMATIC storage class, or options. However, CICS requires that programs be reentrant.

  Although the precompiler uses only the names and data attributes of variable arrays and ignores the alignment, scope, and storage attributes, you should not ignore these restrictions. If you do ignore them, you might have problems compiling the PL/I source code that the precompiler generates.
- You must specify the ALIGNED attribute when you declare varying-length character arrays or varying-length graphic arrays that are to be used in multiple-row INSERT and FETCH statements.

Numeric host-variable arrays

The following diagram shows the syntax for declaring numeric host-variable arrays.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

[2] You can specify the scale for only DECIMAL FIXED.

[3] You can specify host-variable array attributes in any order that is acceptable to PL/I. For example, BIN FIXED(31), BINARY FIXED(31), BIN(31) FIXED, and FIXED BIN(31) are all acceptable.

Example

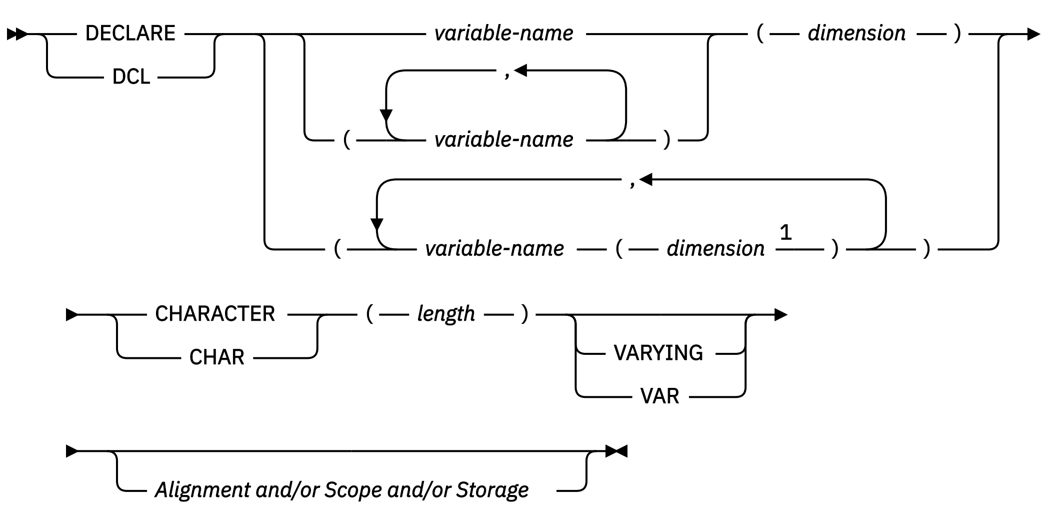
The following example shows a declaration of an indicator array.

```
DCL IND_ARRAY(100) BIN FIXED(15);   /* DCL ARRAY of 100 indicator variables */
```

To use the PL/I decimal floating-point host data types, you need to use the FLOAT(DFP) and ARCH(7) compiler options and the Db2 coprocessor. The maximum precision for extended DECIMAL FLOAT will be 34 (not 33 as it is for hexadecimal float). The maximum precision for short DECIMAL FLOAT will be 7 (not 6 as it is for hexadecimal float).

Character host-variable arrays

The following diagram shows the syntax for declaring character host-variable arrays other than CLOBs.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

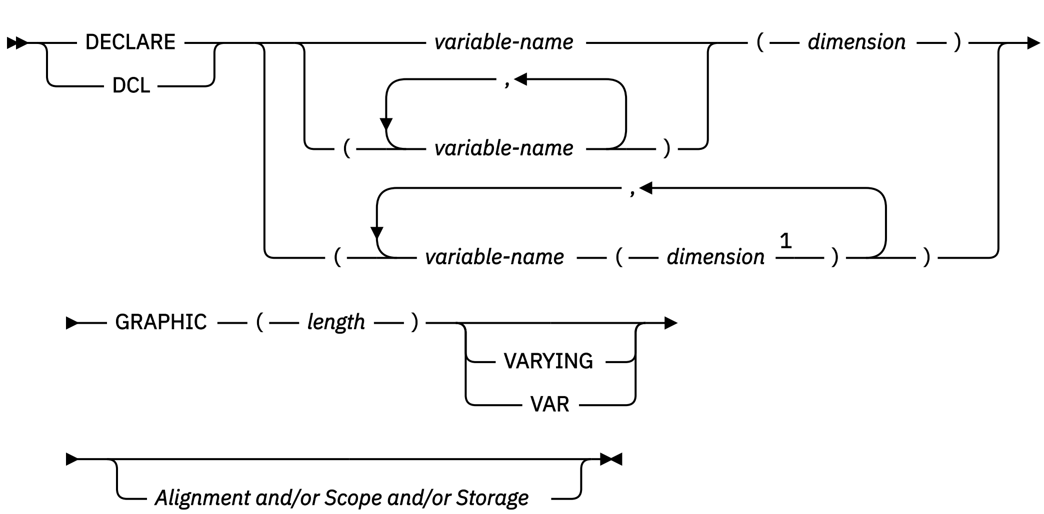
Example

The following example shows the declarations needed to retrieve 10 rows of the department number and name from the department table:

```
DCL DEPTNO(10)    CHAR(3);                     /* Array of ten CHAR(3) variables */
DCL DEPTNAME(10)  CHAR(29) VAR;            /* Array of ten VARCHAR(29) variables */
```

Graphic host-variable arrays

The following diagram shows the syntax for declaring graphic host-variable arrays other than DBCLOBs.

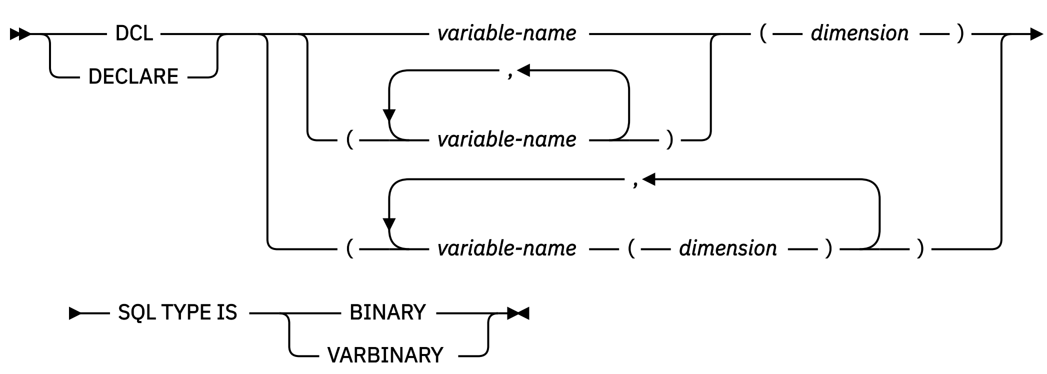


Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

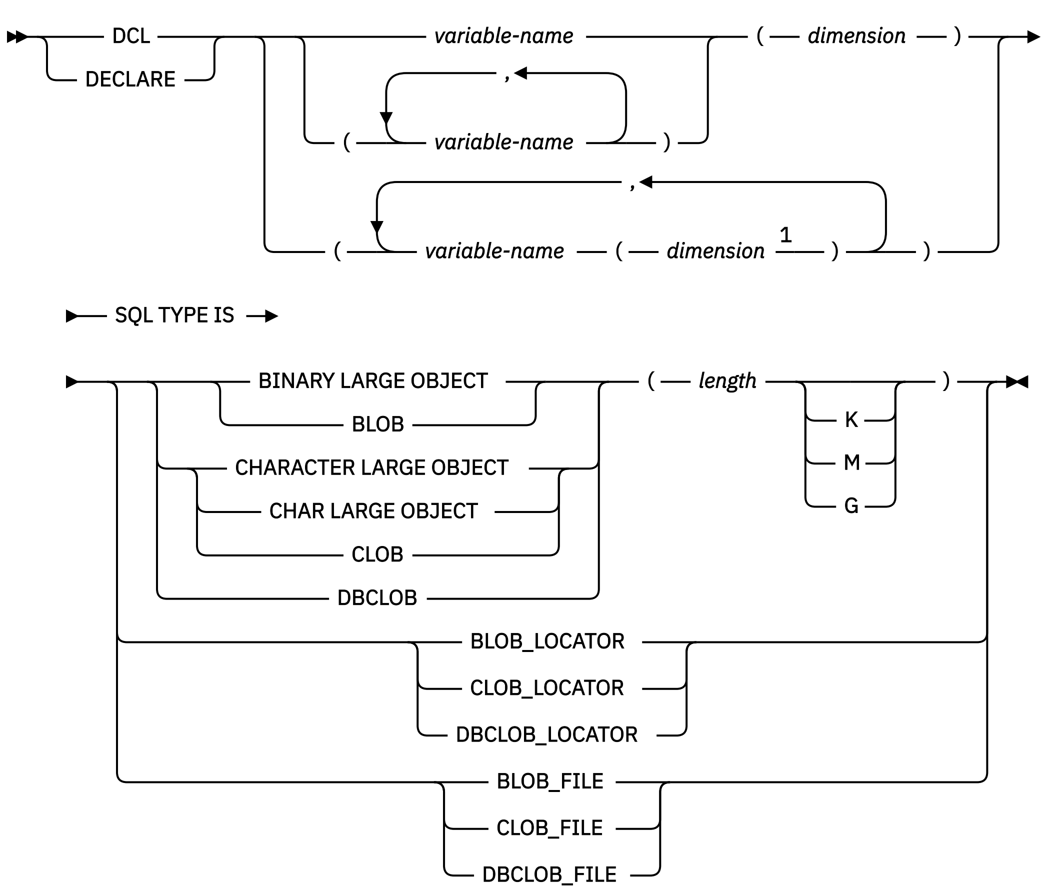
Binary host-variable arrays

The following diagram shows the syntax for declaring binary variable arrays.



LOB, locator, and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host variable, locator, and file reference variable arrays.

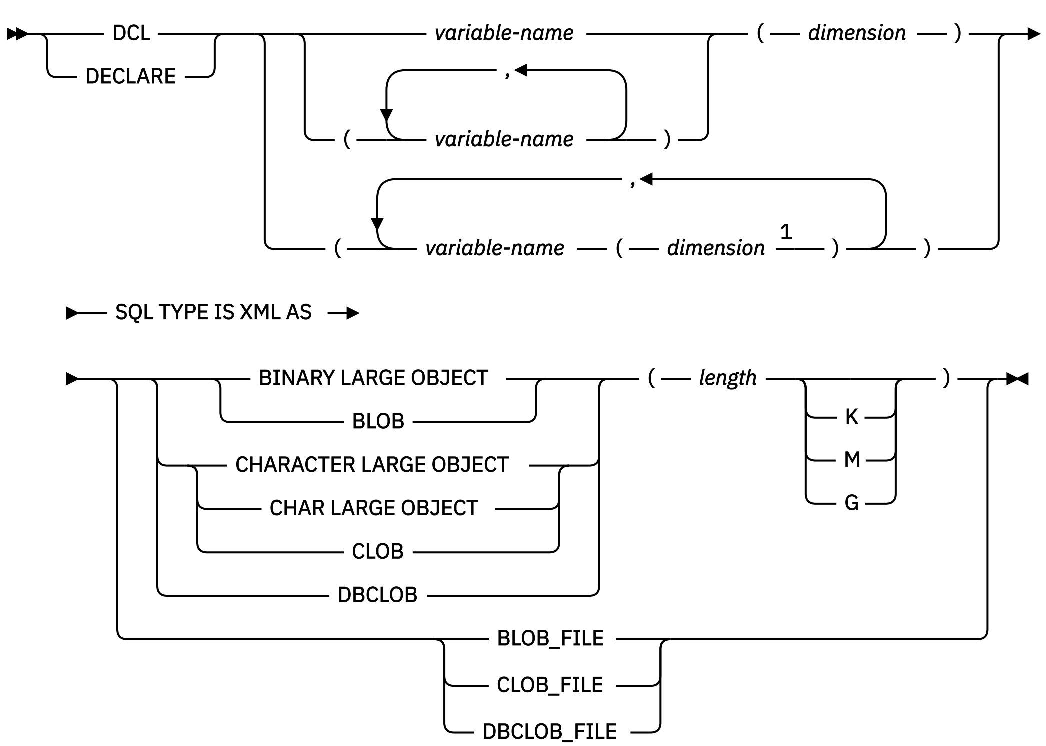


Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

XML host and file reference variable arrays

The following diagram shows the syntax for declaring BLOB, CLOB, and DBCLOB host-variable arrays and file reference variable arrays for XML data types.

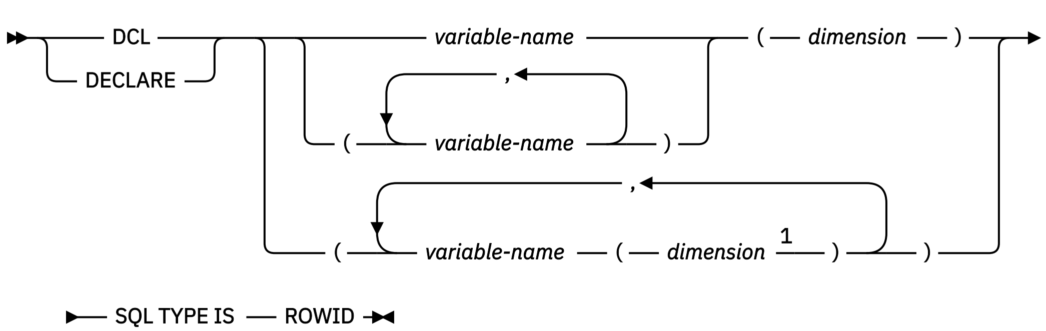


Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

ROWID variable arrays

The following diagram shows the syntax for declaring ROWID variable arrays.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

Related concepts

Using host-variable arrays in SQL statements
Use host-variable arrays in embedded SQL statements to represent values that the program does not know until the query is executed. Host-variable arrays are useful for storing a set of retrieved values or for passing a set of values that are to be inserted into a table.

Host-variable arrays
You can use host-variable arrays to pass a data array between Db2 and your application. A *host-variable array* is a data array that is declared in the host language to be used within an SQL statement.

Host-variable arrays in PL/I, C, C++, and COBOL (Db2 SQL)

Decimal floating-point (DECFLOAT) (Db2 SQL)

Related tasks

Inserting multiple rows of data from host-variable arrays
Use host-variable arrays in your INSERT statement when you do not know at least some of the values to insert until the program runs.

Storing LOB data in a table
Db2 handles LOB data differently than it handles other kinds of data. As a result, in some cases, you need to take additional actions when you define LOB columns and insert the LOB data.

Retrieving multiple rows of data into host-variable arrays
If you know that your query returns multiple rows, you can specify host-variable arrays to store the retrieved column values.

Host structures in PL/I

A PL/I host structure is a structure that contains subordinate levels of scalars. You can use the name of the structure as shorthand notation to reference the list of scalars.

Requirements: Host structure declarations in PL/I must satisfy the following requirements:

- Host structures are limited to two levels.
- You must terminate the host structure variable by ending the declaration with a semicolon, as in the following example:

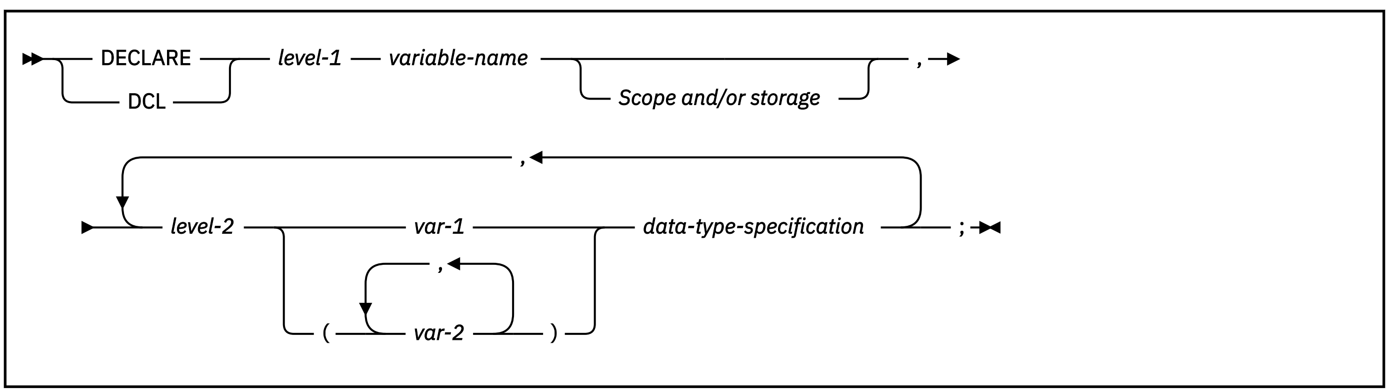
```
DCL 1 A,
    2 B CHAR,
    2 (C, D) CHAR;
DCL (E, F) CHAR;
```

- You can specify host variable attributes in any order that is acceptable to PL/I. For example, BIN FIXED(31), BIN(31) FIXED, and FIXED BIN(31) are all acceptable.

When you reference a host variable, you can qualify it with a structure name. For example, you can specify `STRUCTURE.FIELD`.

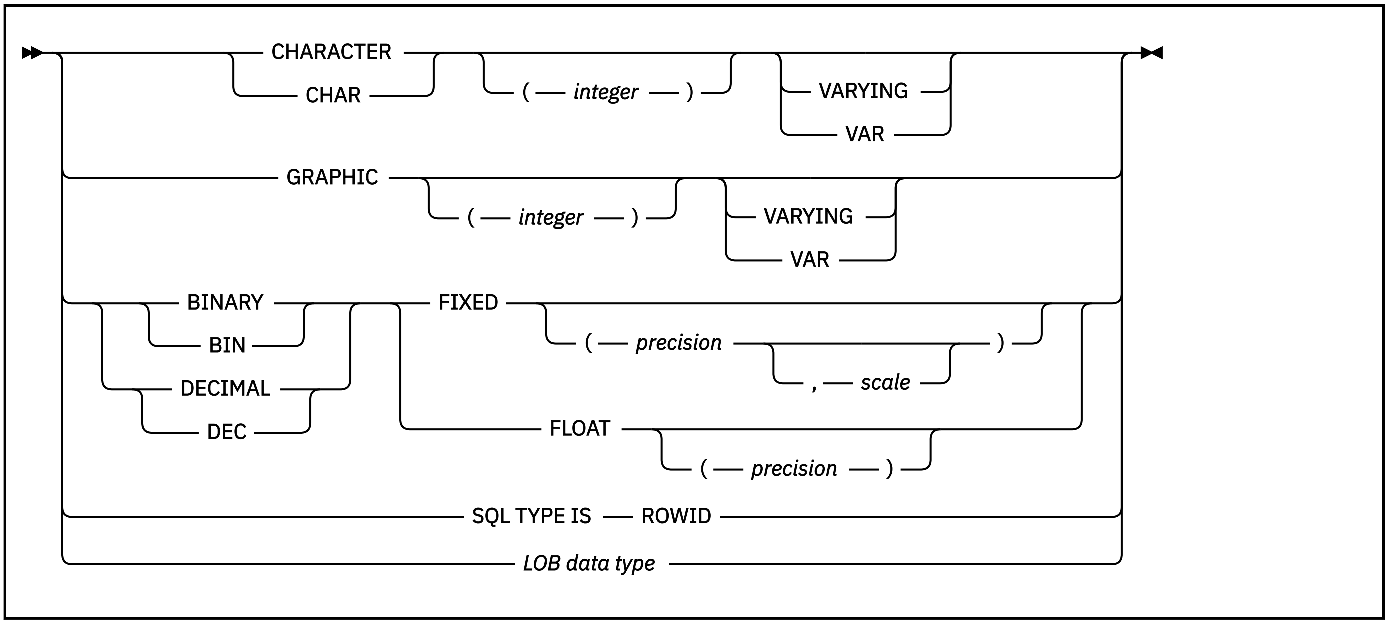
Host structures

The following diagram shows the syntax for declaring host structures.



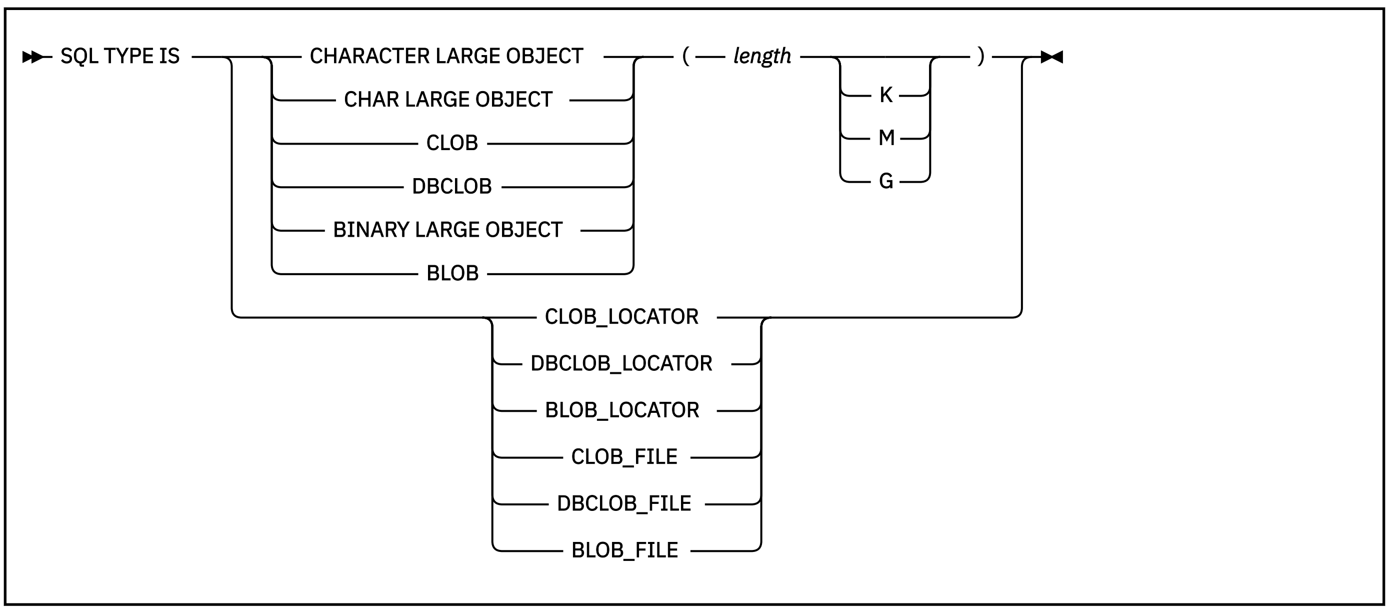
Data types

The following diagram shows the syntax for data types that are used within declarations of host structures.



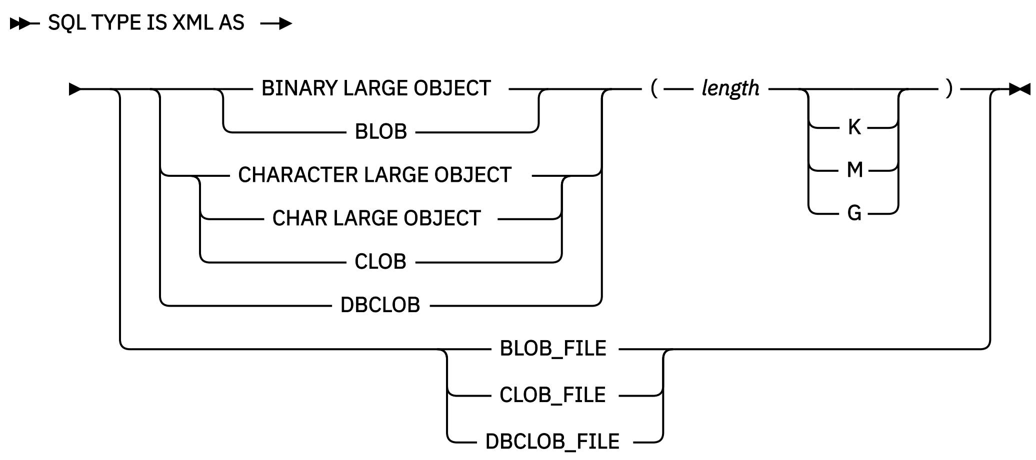
LOB data types

The following diagram shows the syntax for LOB data types that are used within declarations of host structures.



LOB data types for XML data

The following diagram shows the syntax for LOB data types that are used within declarations of host structures for XML data.



Example

In the following example, B is the name of a host structure that contains the scalars C1 and C2.

```
DCL 1 A,
      2 B,
        3 C1 CHAR(...),
        3 C2 CHAR(...);
```

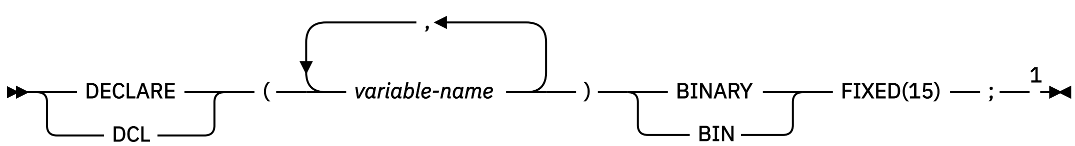
Related concepts

Host structures
Use host structures to pass a group of host variables between Db2 and your application.

Indicator variables in PL/I

An indicator variable is a 2-byte integer (or an integer declared as BIN FIXED(15)). An indicator variable array is an array of 2-byte integers. You declare indicator variables in the same way as host variables. You can mix the declarations of the two types of variables.

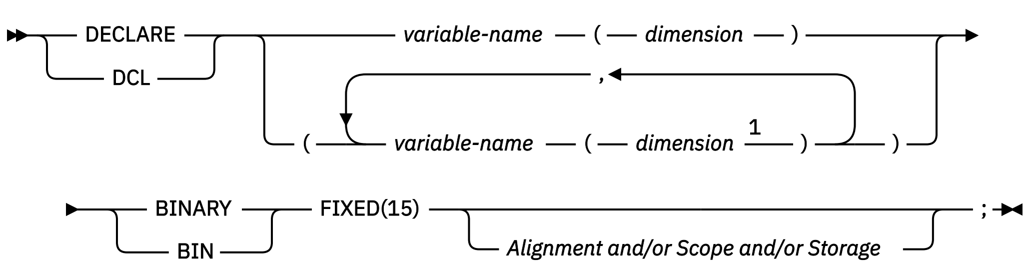
The following diagram shows the syntax for declaring an indicator variable in PL/I.



Notes:

[1] You can specify host variable attributes in any order that is acceptable to PL/I. For example, BIN FIXED(31), BIN(31) FIXED, and FIXED BIN(31) are all acceptable.

The following diagram shows the syntax for declaring an indicator array in PL/I.



Notes:

[1] *dimension* must be an integer constant between 1 and 32767.

Example

The following example shows a FETCH statement with the declarations of the host variables that are needed for the FETCH statement and their associated indicator variables.

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
                               :DAY :DAY_IND,
                               :BGN :BGN_IND,
                               :END :END_IND;
```

You can declare these variables as follows:

```
DCL CLS_CD    CHAR(7);
DCL DAY       BIN FIXED(15);
DCL BGN       CHAR(8);
DCL END       CHAR(8);
DCL (DAY_IND, BGN_IND, END_IND)   BIN FIXED(15);
```

Related concepts

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related tasks

Inserting null values into columns by using indicator variables or arrays
If you need to insert null values into a column, using an indicator variable or array is an easy way to do so. An indicator variable or array is associated with a particular host variable or host-variable array.

Equivalent SQL and PL/I data types

When you declare host variables in your PL/I programs, the precompiler uses equivalent SQL data types. When you retrieve data of a particular SQL data type into a host variable, you need to ensure that the host variable is of an equivalent data type.

The following table describes the SQL data type and the base SQLTYPE and SQLLEN values that the precompiler uses for host variables in SQL statements.

Table 118. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in PL/I programs

PL/I host variable data type	SQLTYPE of host variable "1" on page 724	SQLLEN of host variable	SQL data type
BIN FIXED(*n*) 1≤*n*≤15	500	2	SMALLINT
BIN FIXED(*n*) 16≤*n*≤31	496	4	INTEGER
FIXED BIN(63)	492	8	BIGINT
DEC FIXED(*p*,*s*) 0≤*p*≤31 and 0≤*s*≤*p* "2" on page 724	484	*p* in byte 1, *s* in byte 2	DECIMAL(*p*,*s*)
DEC FLOAT (*p*) where 1 ≤ *p* ≤ 7	996/997	4	DECFLOAT(16) "6" on page 724
DEC FLOAT (*p*) where 8 ≤ *p* ≤ 16	996/997	8	DECFLOAT(16)
DEC FLOAT (*p*) where 17 ≤ *p*	996/997	16	DECFLOAT(34)

Table 118. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in PL/I programs (continued)

PL/I host variable data type	SQLTYPE of host variable "1" on page 724	SQLLEN of host variable	SQL data type
BIN FLOAT(*p*) 1≤*p*≤21	480	4	REAL **or** FLOAT(*n*) 1≤*n*≤21
BIN FLOAT(*p*) 22≤*p*≤53	480	8	DOUBLE PRECISION **or** FLOAT(*n*) 22≤*n*≤53
DEC FLOAT(*m*) 1≤*m*≤6	480	4	FLOAT (single precision)
DEC FLOAT(*m*) 7≤*m*≤16	480	8	FLOAT (double precision)
CHAR(*n*)	452	*n*	CHAR(*n*)
CHAR(*n*) VARYING 1≤*n*≤255	448	*n*	VARCHAR(*n*)
CHAR(*n*) VARYING *n*>255	456	*n*	VARCHAR(*n*)
GRAPHIC(*n*)	468	*n*	GRAPHIC(*n*)
GRAPHIC VARYING(*n*)	464	*n*	VARGRAPHIC(*n*)
SQL TYPE IS BINARY(*n*), 1≤*n*≤255	912	*n*	BINARY(*n*)
SQL TYPE IS VARBINARY(*n*), 1≤*n*≤32704	908	*n*	VARBINARY(*n*)
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Result set locator "3" on page 724
SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	976	4	Table locator "3" on page 724
SQL TYPE IS BLOB_LOCATOR	960	4	BLOB locator "3" on page 724
SQL TYPE IS CLOB_LOCATOR	964	4	CLOB locator "3" on page 724
SQL TYPE IS DBCLOB_LOCATOR	968	4	DBCLOB locator "3" on page 724
SQL TYPE IS BLOB(*n*) 1≤*n*≤2147483647	404	*n*	BLOB(*n*)
SQL TYPE IS CLOB(*n*) 1≤*n*≤2147483647	408	*n*	CLOB(*n*)
SQL TYPE IS DBCLOB(*n*) 1≤*n*≤1073741823 "4" on page 724	412	*n*	DBCLOB(*n*) "4" on page 724
SQL TYPE IS XML AS BLOB(*n*)	404	0	XML
SQL TYPE IS XML AS CLOB(*n*)	408	0	XML
SQL TYPE IS XML AS DBCLOB(*n*)	412	0	XML
SQL TYPE IS BLOB_FILE	916/917	267	BLOB file reference "3" on page 724
SQL TYPE IS CLOB_FILE	920/921	267	CLOB file reference "3" on page 724

Table 118. SQL data types, SQLLEN values, and SQLTYPE values that the precompiler uses for host variables in PL/I programs (continued)

PL/I host variable data type	SQLTYPE of host variable "1" on page 724	SQLLEN of host variable	SQL data type
SQL TYPE IS DBCLOB_FILE	924/925	267	DBCLOB file reference "3" on page 724
SQL TYPE IS XML AS BLOB_FILE	916/917	267	XML BLOB file reference "3" on page 724
SQL TYPE IS XML AS CLOB_FILE	920/921	267	XML CLOB file reference "3" on page 724
SQL TYPE IS XML AS DBCLOB_FILE	924/925	267	XML DBCLOB file reference "3" on page 724
SQL TYPE IS ROWID	904	40	ROWID
WIDECHAR(*n*)	468	*n*	GRAPHIC(*n*) "5" on page 724
WIDECHAR VARYING(*n*)	464	*n*	VARGRAPHIC(*n*) "5" on page 724

The following table shows equivalent PL/I host variables for each SQL data type. Use this table to determine the PL/I data type for host variables that you define to receive output from the database. For example, if you retrieve TIMESTAMP data, you can define a variable of type CHAR(*n*).

This table shows direct conversions between SQL data types and PL/I data types. However, a number of SQL data types are compatible. When you do assignments or comparisons of data that have compatible data types, Db2 converts those compatible data types.

Table 119. PL/I host variable equivalents that you can use when retrieving data of a particular SQL data type

SQL data type	PL/I host variable equivalent	Remarks
SMALLINT	BIN FIXED(*n*)	1≤*n*≤15
INTEGER	BIN FIXED(*n*)	16≤*n*≤31
BIGINT	FIXED BIN(63)	"7" on page 724
DECIMAL(*p*,*s*) or NUMERIC(*p*,*s*)	If *p*<16: DEC FIXED(*p*) or DEC FIXED(*p*,*s*)	*p* is precision; *s* is scale. 1≤*p*≤31 and 0≤*s*≤*p* If *p*>15, the PL/I compiler must support 31-digit decimal variables.
DECFLOAT(16)	DEC FLOAT (*p*)	1 ≤ *p* ≤ 7
DECFLOAT(16)	DEC FLOAT (*p*)	8 ≤ *p* ≤ 16
DECFLOAT(34)	DEC FLOAT (*p*)	17 ≤ *p*
REAL or FLOAT(*n*)	BIN FLOAT(*p*) or DEC FLOAT(*m*)	1≤*n*≤21, 1≤*p*≤21, and 1≤*m*≤6
DOUBLE PRECISION, DOUBLE, or FLOAT(*n*)	BIN FLOAT(*p*) or DEC FLOAT(*m*)	22≤*n*≤53, 22≤*p*≤53, and 7≤*m*≤16
CHAR(*n*)	CHAR(*n*)	1≤*n*≤255
VARCHAR(*n*)	CHAR(*n*) VAR	
GRAPHIC(*n*)	GRAPHIC(*n*) or WIDECHAR(*n*) "2" on page 724	*n* refers to the number of double-byte characters, not to the number of bytes.

Table 119. PL/I host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	PL/I host variable equivalent	Remarks
VARGRAPHIC(*n*)	GRAPHIC(*n*) VARYING or WIDECHAR(*n*) VARYING	*n* refers to the number of double-byte characters, not to the number of bytes.
BINARY(*n*)	SQL TYPE IS BINARY(*n*)	1≤*n*≤255
VARBINARY(*n*)	SQL TYPE IS VARBINARY(*n*)	1≤*n*≤32704
DATE	CHAR(*n*)	If you are using a date exit routine, that routine determines *n*; otherwise, *n* must be at least 10.
TIME	CHAR(*n*)	If you are using a time exit routine, that routine determines *n*. Otherwise, *n* must be at least 6; to include seconds, *n* must be at least 8.
TIMESTAMP	CHAR(*n*)	*n* must be at least 19. To include microseconds, *n* must be 26; if *n* is less than 26, the microseconds part is truncated.
TIMESTAMP(0)	CHAR(*n*)	*n* must be at least 19.
TIMESTAMP(*p*) *p* > 0	CHAR(*n*)	*n* must be at least 19. To include fractional seconds, *n* must be 20+*x* where *x* is the number of fractional seconds to include; if *x* is less than *p*, truncation occurs on the fractional seconds part.
TIMESTAMP(0) WITH TIME ZONE	CHAR(*n*) VAR	*n* must be at least 25.
TIMESTAMP(*p*) WITH TIME ZONE	CHAR(*n*) VAR	*n* must be at least 26+*p*.
Result set locator	SQL TYPE IS RESULT_SET_LOCATOR	Use this data type only for receiving result sets. "3" on page 724
Table locator	SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. "3" on page 724
BLOB locator	SQL TYPE IS BLOB_LOCATOR	Use this data type only to manipulate data in BLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
CLOB locator	SQL TYPE IS CLOB_LOCATOR	Use this data type only to manipulate data in CLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
DBCLOB locator	SQL TYPE IS DBCLOB_LOCATOR	Use this data type only to manipulate data in DBCLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
BLOB(*n*)	SQL TYPE IS BLOB(*n*)	1≤n≤2147483647 "6" on page 724, "7" on page 724

Table 119. PL/I host variable equivalents that you can use when retrieving data of a particular SQL data type (continued)

SQL data type	PL/I host variable equivalent	Remarks
CLOB(*n*)	SQL TYPE IS CLOB(*n*)	1≤n≤2147483647 "6" on page 724, "7" on page 724
DBCLOB(*n*)	SQL TYPE IS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤n≤1073741823 "5" on page 724, "7" on page 724
XML	SQL TYPE IS XML AS BLOB(*n*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS CLOB(*n*)	1≤n≤2147483647
XML	SQL TYPE IS XML AS DBCLOB(*n*)	*n* is the number of double-byte characters. 1≤n≤1073741823 "6" on page 724
BLOB file reference	SQL TYPE IS BLOB_FILE	Use this data type only to manipulate data in BLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
CLOB file reference	SQL TYPE IS CLOB_FILE	Use this data type only to manipulate data in CLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
DBCLOB file reference	SQL TYPE IS DBCLOB_FILE	Use this data type only to manipulate data in DBCLOB columns. "3" on page 724, "6" on page 724, "7" on page 724
XML BLOB file reference	SQL TYPE IS XML AS BLOB_FILE	Use this data type only to manipulate XML data as BLOB files. "3" on page 724
XML CLOB file reference	SQL TYPE IS XML AS CLOB_FILE	Use this data type only to manipulate XML data as CLOB files. "3" on page 724
XML DBCLOB file reference	SQL TYPE IS XML AS DBCLOB_FILE	Use this data type only to manipulate XML data as DBCLOB files. "3" on page 724
ROWID	SQL TYPE IS ROWID	

Table notes:

The following notes apply as indicated to Table 118 on page 720 and Table 119 on page 722.

1. If a host variable includes an indicator variable, the SQLTYPE value is the base SQLTYPE value plus 1.
2. If *p*=0, Db2 interprets it as DECIMAL(31). For example, Db2 interprets a PL/I data type of DEC FIXED(0,0) to be DECIMAL(31,0), which equates to the SQL data type of DECIMAL(31,0).
3. Do not use this data type as a column type.
4. *n* is the number of double-byte characters.
5. CCSID 1200 is always assigned to WIDECHAR type host var.
6. The data type conversions can be used only if the Db2 coprocessor is used, and the PL/I compiler options FLOAT(DFP) and ARCH(7) are specified.
7. Specify the following compiler options when you compile your program: LIMITS(FIXEDBIN(63), FIXEDDEC(31)).

Related concepts

Compatibility of SQL and language data types

The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

LOB host variable, LOB locator, and LOB file reference variable declarations
When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator. Alternatively, you need to declare LOB file reference variables to point to the LOB data.

Host variable data types for XML data in embedded SQL applications (Db2 Programming for XML)

REXX applications that issue SQL statements

You can code SQL statements in a REXX programs wherever you can use REXX commands.

Db2 REXX Language Support supports all dynamic SQL statements and the following static SQL statements:

- CALL
- CLOSE
- CONNECT
- DECLARE CURSOR
- DESCRIBE *prepared statement or table*
- DESCRIBE CURSOR
- DESCRIBE INPUT
- DESCRIBE PROCEDURE
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- OPEN
- PREPARE
- RELEASE *connection*
- SET CONNECTION
- SET CURRENT PACKAGE PATH
- SET CURRENT PACKAGESET
- SET *host-variable* = CURRENT DATE
- SET *host-variable* = CURRENT DEGREE
- SET *host-variable* = CURRENT MEMBER
- SET *host-variable* = CURRENT PACKAGESET
- SET *host-variable* = CURRENT PATH
- SET *host-variable* = CURRENT SERVER
- SET *host-variable* = CURRENT SQLID
- SET *host-variable* = CURRENT TIME
- SET *host-variable* = CURRENT TIMESTAMP
- SET *host-variable* = CURRENT TIMEZONE

Each SQL statement in a REXX program must begin with EXECSQL, in either upper-, lower-, or mixed-case. One of the following items must follow EXECSQL:

- An SQL statement enclosed in single or double quotation marks.
- A REXX variable that contains an SQL statement. The REXX variable must not be preceded by a colon.

For example, you can use either of the following methods to execute the COMMIT statement in a REXX program:

```
EXECSQL "COMMIT"
```

```
rexxvar="COMMIT"
EXECSQL rexxvar
```

The following dynamic statements must be executed using EXECUTE IMMEDIATE or PREPARE and EXECUTE under DSNREXX:

- DECLARE GLOBAL TEMPORARY TABLE
- SET CURRENT DEBUG MODE
- SET CURRENT DECFLOAT ROUNDING MODE
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT QUERY ACCELERATION
- SET CURRENT REFRESH AGE
- SET CURRENT ROUTINE VERSION
- SET SCHEMA

You cannot execute a SELECT, INSERT, UPDATE, MERGE, or DELETE statement that contains host variables. Instead, you must execute PREPARE on the statement, with parameter markers substituted for the host variables, and then use the host variables in an EXECUTE, OPEN, or FETCH statement. See "Host variables" on page 477 for more information.

An SQL statement follows rules that apply to REXX commands. The SQL statement can optionally end with a semicolon and can be enclosed in single or double quotation marks, as in the following example:

```
'EXECSQL COMMIT';
```

Comments
You cannot include REXX comments (/* ... */) or SQL comments (--) within SQL statements. However, you can include REXX comments anywhere else in the program.

Delimiters for SQL statements
Delimit SQL statements in REXX program by preceding the statement with EXECSQL. If the statement is in a literal string, enclose it in single or double quotation marks.

Continuation for SQL statements
SQL statements that span lines follow REXX rules for statement continuation. You can break the statement into several strings, each of which fits on a line, and separate the strings with commas or with concatenation operators followed by commas. For example, either of the following statements is valid:

```
EXECSQL ,
   "UPDATE DSN8B10.DEPT" ,
   "SET MGRNO = '000010'" ,
   "WHERE DEPTNO = 'D11'"
```

```
"EXECSQL " || ,
"  UPDATE DSN8B10.DEPT " || ,
"  SET MGRNO = '000010'" || ,
"  WHERE DEPTNO = 'D11'"
```

Including code
The EXECSQL INCLUDE statement is not valid for REXX. You therefore cannot include externally defined SQL statements in a program.

Margins
Like REXX commands, SQL statements can begin and end anywhere on a line.

You can use any valid REXX name that does not end with a period as a host variable. However, host variable names should not begin with 'SQL', 'RDI', 'DSN', 'RXSQL', or 'QRW'. Variable names can be at most 64 bytes.

Nulls
A REXX null value and an SQL null value are different. The REXX language has a null string (a string of length 0) and a null clause (a clause that contains only blanks and comments). The SQL null value is a special value that is distinct from all nonnull values and denotes the absence of a value. Assigning a REXX null value to a Db2 column does not make the column value null.

Statement labels
You can precede an SQL statement with a label, in the same way that you label REXX commands.

Handling SQL error codes
Rexx applications can request more information about SQL errors from Db2. For more information, see “Handling SQL error codes in REXX applications” on page 750.

Related tasks

Overview of programming applications that access Db2 for z/OS data
Applications that interact with Db2 must first connect to Db2. They can then read, add, or modify data or manipulate Db2 objects.

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

REXX programming examples

You can write Db2 programs in REXX. These programs can access a local or remote Db2 subsystem and can execute static or dynamic SQL statements. This information contains several such programming examples.

To prepare and run these applications, use the JCL in *prefix*.SDSNSAMP as a model for your JCL.

Related reference

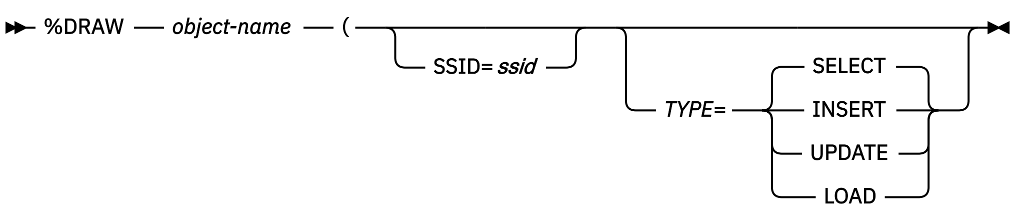
Assembler, C, C++, COBOL, PL/I, and REXX programming examples (Db2 Programming samples)
Db2 for z/OS Exchange

Sample Db2 REXX application

You can use a REXX application to accept a table name as input and produce a SELECT, INSERT, or UPDATE SQL statement or a LOAD utility statement for the specified table as output.

The following example shows a complete Db2 REXX application named DRAW. DRAW must be invoked from the command line of an ISPF edit session. DRAW takes a table or view name as input and produces a SELECT, INSERT, or UPDATE SQL statement or a LOAD utility control statement that includes the columns of the table as output.

DRAW syntax:



DRAW parameters:

object-name
The name of the table or view for which DRAW builds an SQL statement or utility control statement. The name can be a one-, two-, or three-part name. The table or view to which *object-name* refers must exist before DRAW can run.

object-name is a required parameter.

SSID=*ssid*
Specifies the name of the local Db2 subsystem.

S can be used as an abbreviation for SSID.

If you invoke DRAW from the command line of the edit session in SPUFI, SSID=*ssid* is an optional parameter. DRAW uses the subsystem ID from the DB2I Defaults panel.

TYPE=*operation-type*
The type of statement that DRAW builds.

T can be used as an abbreviation for TYPE.

operation-type has one of the following values:

SELECT
Builds a SELECT statement in which the result table contains all columns of *object-name*.

S can be used as an abbreviation for SELECT.

INSERT
Builds a template for an INSERT statement that inserts values into all columns of *object-name*. The template contains comments that indicate where the user can place column values.

I can be used as an abbreviation for INSERT.

UPDATE
Builds a template for an UPDATE statement that updates columns of *object-name*. The template contains comments that indicate where the user can place column values and qualify the update operation for selected rows.

U can be used as an abbreviation for UPDATE.

LOAD
Builds a template for a LOAD utility control statement for *object-name*.

L can be used as an abbreviation for LOAD.

TYPE=*operation-type* is an optional parameter. The default is TYPE=SELECT.

DRAW data sets:

Edit data set
The data set from which you issue the DRAW command when you are in an ISPF edit session. If you issue the DRAW command from a SPUFI session, this data set is the data set that you specify in field 1 of the main SPUFI panel (DSNESP01). The output from the DRAW command goes into this data set.

DRAW return codes:

Return code
Meaning

0
Successful completion.

12
An error occurred when DRAW edited the input file.

20
One of the following errors occurred:

- No input parameters were specified.
- One of the input parameters was not valid.

• An SQL error occurred when the output statement was generated.

Examples of DRAW invocation:

Generate a SELECT statement for table DSN8B10.EMP at the local subsystem. Use the default DB2I subsystem ID.

The DRAW invocation is:

```
DRAW DSN8B10.EMP (TYPE=SELECT
```

The output is:

```
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
  "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
  "SALARY" , "BONUS" , "COMM"
FROM DSN8B10.EMP
```

Generate a template for an INSERT statement that inserts values into table DSN8B10.EMP at location SAN_JOSE. The local subsystem ID is DSN.

The DRAW invocation is:

```
DRAW SAN_JOSE.DSN8B10.EMP (TYPE=INSERT SSID=DSN
```

The output is:

```
INSERT INTO SAN_JOSE.DSN8B10.EMP  ( "EMPNO" , "FIRSTNME" , "MIDINIT" ,
  "LASTNAME" ,  "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" ,
  "EDLEVEL" , "SEX" , "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
   VALUES (
-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE
                         , -- EMPNO              CHAR(6) NOT NULL
                         , -- FIRSTNME           VARCHAR(12) NOT NULL
                         , -- MIDINIT            CHAR(1) NOT NULL
                         , -- LASTNAME           VARCHAR(15) NOT NULL
                         , -- WORKDEPT           CHAR(3)
                         , -- PHONENO            CHAR(4)
                         , -- HIREDATE           DATE
                         , -- JOB                CHAR(8)
                         , -- EDLEVEL            SMALLINT
                         , -- SEX                CHAR(1)
                         , -- BIRTHDATE          DATE
                         , -- SALARY             DECIMAL(9,2)
                         , -- BONUS              DECIMAL(9,2)
                         ) -- COMM               DECIMAL(9,2)
```

Generate a template for an UPDATE statement that updates values of table DSN8B10.EMP. The local subsystem ID is DSN.

The DRAW invocation is:

```
DRAW DSN8B10.EMP (TYPE=UPDATE SSID=DSN
```

The output is:

```
UPDATE DSN8B10.EMP  SET
-- COLUMN NAME           ENTER VALUES BELOW      DATA TYPE
   "EMPNO"=                                   -- CHAR(6) NOT NULL
 , "FIRSTNME"=                                -- VARCHAR(12) NOT NULL
 , "MIDINIT"=                                 -- CHAR(1) NOT NULL
 , "LASTNAME"=                                -- VARCHAR(15) NOT NULL
 , "WORKDEPT"=                                -- CHAR(3)
 , "PHONENO"=                                 -- CHAR(4)
 , "HIREDATE"=                                -- DATE
 , "JOB"=                                     -- CHAR(8)
 , "EDLEVEL"=                                 -- SMALLINT
 , "SEX"=                                     -- CHAR(1)
 , "BIRTHDATE"=                               -- DATE
 , "SALARY"=                                  -- DECIMAL(9,2)
 , "BONUS"=                                   -- DECIMAL(9,2)
```

```
 , "COMM"=                                    -- DECIMAL(9,2)
WHERE
```

Generate a LOAD control statement to load values into table DSN8B10.EMP. The local subsystem ID is DSN.

The draw invocation is:

```
DRAW DSN8B10.EMP (TYPE=LOAD SSID=DSN
```

The output is:

```
LOAD DATA INDDN SYSREC INTO TABLE DSN8B10.EMP
 ( "EMPNO"              POSITION(    1) CHAR(6)
 , "FIRSTNME"           POSITION(    8) VARCHAR
 , "MIDINIT"            POSITION(   21) CHAR(1)
 , "LASTNAME"           POSITION(   23) VARCHAR
 , "WORKDEPT"           POSITION(   39) CHAR(3)
                          NULLIF(   39)='?'
 , "PHONENO"            POSITION(   43) CHAR(4)
                          NULLIF(   43)='?'
 , "HIREDATE"           POSITION(   48) DATE EXTERNAL
                          NULLIF(   48)='?'
 , "JOB"                POSITION(   59) CHAR(8)
                          NULLIF(   59)='?'
 , "EDLEVEL"            POSITION(   68) SMALLINT
                          NULLIF(   68)='?'
 , "SEX"                POSITION(   71) CHAR(1)
                          NULLIF(   71)='?'
 , "BIRTHDATE"          POSITION(   73) DATE EXTERNAL
                          NULLIF(   73)='?'
 , "SALARY"             POSITION(   84) DECIMAL EXTERNAL(9,2)
                          NULLIF(   84)='?'
 , "BONUS"              POSITION(   90) DECIMAL EXTERNAL(9,2)
                          NULLIF(   90)='?'
 , "COMM"               POSITION(   96) DECIMAL EXTERNAL(9,2)
                          NULLIF(   96)='?'
 )
```

DRAW source code:

```
/* REXX ***************************************************************/
L1 = WHEREAMI()
/*
DRAW creates basic SQL queries by retrieving the description of a
table.  You must specify the name of the table or view to be queried.
You can specify the type of query you want to compose.  You might need
to specify the name of the DB2 subsystem.
>>--DRAW-----tablename-----|--------------------------|------->< 
                           |-(-|-Ssid=subsystem-name-|-|
                               |       +-Select-+    |
                               |-Type=-|-Insert-|----|
                                       |-Update-|
                                       +--Load--+
Ssid=subsystem-name
    subsystem-name specified the name of a DB2 subsystem.
Select
    Composes a basic query for selecting data from the columns of a
    table or view.  If TYPE is not specified, SELECT is assumed.
    Using SELECT with the DRAW command produces a query that would
    retrieve all rows and all columns from the specified table.  You
    can then modify the query as needed.
    A SELECT query of EMP composed by DRAW looks like this:
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM DSN8B10.EMP
            If you include a location qualifier, the query looks like this:
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM STLEC1.DSN8B10.EMP
```

```
    To use this SELECT query, type the other clauses you need.  If
    you are selecting from more than one table, use a DRAW command
    for each table name you want represented.
```

```
Insert
    Composes a basic query to insert data into the columns of a table
    or view.
    The following example shows an INSERT query of EMP that
    DRAW composed:
INSERT INTO DSN8B10.EMP  ( "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" ,
       "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" ,
       "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
   VALUES (
-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE
                         , -- EMPNO              CHAR(6) NOT NULL
                         , -- FIRSTNME           VARCHAR(12) NOT NULL
                         , -- MIDINIT            CHAR(1) NOT NULL
                         , -- LASTNAME           VARCHAR(15) NOT NULL
                         , -- WORKDEPT           CHAR(3)
                         , -- PHONENO            CHAR(4)
                         , -- HIREDATE           DATE
                         , -- JOB                CHAR(8)
                         , -- EDLEVEL            SMALLINT
                         , -- SEX                CHAR(1)
                         , -- BIRTHDATE          DATE
                         , -- SALARY             DECIMAL(9,2)
                         , -- BONUS              DECIMAL(9,2)
                         ) -- COMM               DECIMAL(9,2)
    To insert values into EMP, type values to the left of the
    column names.
Update
    Composes a basic query to change the data in a table or view.
    The following example shows an UPDATE query of EMP composed
    by DRAW:
```

```
UPDATE DSN8B10.EMP  SET
-- COLUMN NAME          ENTER VALUES BELOW      DATA TYPE
   "EMPNO"=                                  -- CHAR(6) NOT NULL
 , "FIRSTNME"=                               -- VARCHAR(12) NOT NULL
 , "MIDINIT"=                                -- CHAR(1) NOT NULL
 , "LASTNAME"=                               -- VARCHAR(15) NOT NULL
 , "WORKDEPT"=                               -- CHAR(3)
 , "PHONENO"=                                -- CHAR(4)
 , "HIREDATE"=                               -- DATE
 , "JOB"=                                    -- CHAR(8)
 , "EDLEVEL"=                                -- SMALLINT
 , "SEX"=                                    -- CHAR(1)
 , "BIRTHDATE"=                              -- DATE
 , "SALARY"=                                 -- DECIMAL(9,2)
 , "BONUS"=                                  -- DECIMAL(9,2)
 , "COMM"=                                   -- DECIMAL(9,2)
WHERE
    To use this UPDATE query, type the changes you want to make to
    the right of the column names, and delete the lines you do not
    need.  Be sure to complete the WHERE clause.
Load
    Composes a load statement to load the data in a table.
    The following example shows a LOAD statement of EMP composed
    by DRAW:
LOAD DATA INDDN SYSREC INTO TABLE DSN8B10 .EMP
 ( "EMPNO"              POSITION(    1) CHAR(6)
 , "FIRSTNME"           POSITION(    8) VARCHAR
 , "MIDINIT"            POSITION(   21) CHAR(1)
 , "LASTNAME"           POSITION(   23) VARCHAR
 , "WORKDEPT"           POSITION(   39) CHAR(3)
                          NULLIF(   39)='?'
 , "PHONENO"            POSITION(   43) CHAR(4)
                          NULLIF(   43)='?'
 , "HIREDATE"           POSITION(   48) DATE EXTERNAL
                          NULLIF(   48)='?'
 , "JOB"                POSITION(   59) CHAR(8)
                          NULLIF(   59)='?'
 , "EDLEVEL"            POSITION(   68) SMALLINT
                          NULLIF(   68)='?'
 , "SEX"                POSITION(   71) CHAR(1)
                          NULLIF(   71)='?'
 , "BIRTHDATE"          POSITION(   73) DATE EXTERNAL
                          NULLIF(   73)='?'
 , "SALARY"             POSITION(   84) DECIMAL EXTERNAL(9,2)
                          NULLIF(   84)='?'
 , "BONUS"              POSITION(   90) DECIMAL EXTERNAL(9,2)
                          NULLIF(   90)='?'
 , "COMM"               POSITION(   96) DECIMAL EXTERNAL(9,2)
```

```
                          NULLIF(   96)='?'
 )
```

```
    To use this LOAD statement, type the changes you want to make,
    and delete the lines you do not need.
*/
   L2 = WHEREAMI()
/**********************************************************************/
/* TRACE ?R                                                           */
/**********************************************************************/
Address ISPEXEC
"ISREDIT MACRO (ARGS) NOPROCESS"
If ARGS = "" Then
Do
  Do I = L1+2 To L2-2;Say SourceLine(I);End
  Exit (20)
End
Parse Upper Var Args Table "(" Parms
Parms = Translate(Parms," ",",")
Type = "SELECT" /* Default */
SSID = ""       /* Default */
"VGET (DSNEOV01)"
If RC = 0 Then SSID = DSNEOV01
If (Parms <> "") Then
Do Until(Parms = "")
Parse Var Parms Var "=" Value Parms
  If Var = "T" | Var = "TYPE" Then Type = Value
  Else
  If Var = "S" | Var = "SSID" Then SSID = Value
  Else
    Exit (20)
End
"CONTROL ERRORS RETURN"
"ISREDIT (LEFTBND,RIGHTBND) = BOUNDS"
"ISREDIT (LRECL) = DATA_WIDTH" /*LRECL*/
BndSize = RightBnd - LeftBnd + 1
If BndSize > 72 Then BndSize = 72
"ISREDIT PROCESS DEST"
Select
  When rc = 0 Then
    'ISREDIT (ZDEST) = LINENUM .ZDEST'
  When rc <= 8 Then /* No A or B entered */
    Do
       zedsmsg = 'Enter "A"/"B" line cmd'
       zedlmsg = 'DRAW requires an "A" or "B" line command'
       'SETMSG MSG(ISRZ001)'
       Exit 12
    End
  When rc < 20 Then /* Conflicting line commands - edit sets message */
    Exit 12
  When rc = 20 Then
    zdest = 0
  Otherwise
    Exit 12
End
```

```
SQLTYPE. = "UNKNOWN TYPE"
VCHTYPE  = 448; SQLTYPES.VCHTYPE  = 'VARCHAR'
CHTYPE   = 452; SQLTYPES.CHTYPE   = 'CHAR'
LVCHTYPE = 456; SQLTYPES.LVCHTYPE = 'VARCHAR'
VGRTYP   = 464; SQLTYPES.VGRTYP   = 'VARGRAPHIC'
GRTYP    = 468; SQLTYPES.GRTYP    = 'GRAPHIC'
LVGRTYP  = 472; SQLTYPES.LVGRTYP  = 'VARGRAPHIC'
FLOTYPE  = 480; SQLTYPES.FLOTYPE  = 'FLOAT'
DCTYPE   = 484; SQLTYPES.DCTYPE   = 'DECIMAL'
INTYPE   = 496; SQLTYPES.INTYPE   = 'INTEGER'
SMTYPE   = 500; SQLTYPES.SMTYPE   = 'SMALLINT'
DATYPE   = 384; SQLTYPES.DATYPE   = 'DATE'
TITYPE   = 388; SQLTYPES.TITYPE   = 'TIME'
TSTYPE   = 392; SQLTYPES.TSTYPE   = 'TIMESTAMP'
Address TSO "SUBCOM DSNREXX"           /* HOST CMD ENV AVAILABLE? */
IF RC THEN                             /* NO, LET'S MAKE ONE      */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX') /* ADD HOST CMD ENV  */
Address DSNREXX "CONNECT" SSID
If SQLCODE ^= 0 Then Call SQLCA
Address DSNREXX "EXECSQL DESCRIBE TABLE :TABLE INTO :SQLDA"
If SQLCODE ^= 0 Then Call SQLCA
Address DSNREXX "EXECSQL COMMIT"
Address DSNREXX "DISCONNECT"
If SQLCODE ^= 0 Then Call SQLCA
```

```
Select
  When (Left(Type,1) = "S") Then
    Call DrawSelect
  When (Left(Type,1) = "I") Then
    Call DrawInsert
  When (Left(Type,1) = "U") Then
    Call DrawUpdate
  When (Left(Type,1) = "L") Then
    Call DrawLoad
  Otherwise EXIT (20)
End
Do I = LINE.0 To 1 By -1
  LINE = COPIES(" ",LEFTBND-1)||LINE.I
  'ISREDIT LINE_AFTER 'zdest' = DATALINE (Line)'
End
line1 = zdest + 1
'ISREDIT CURSOR = 'line1 0
Exit
```

```
/*********************************************************************/
WHEREAMI:; RETURN SIGL
/*********************************************************************/
/* Draw SELECT                                                       */
/*********************************************************************/
DrawSelect:
  Line.0 = 0
  Line = "SELECT"
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = '"'SQLDA.I.SQLNAME'"'
    Null = SQLDA.I.SQLTYPE//2
    If Length(Line)+Length(ColName)+LENGTH(" ,") > BndSize THEN
    Do
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
      Line = "      "
    End
    Line = Line ColName
  End I
  If Line ^= "" Then
  Do
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
    Line = "      "
  End
  L = Line.0 + 1; Line.0 = L
  Line.L = "FROM" TABLE
  Return
/*********************************************************************/
/* Draw INSERT                                                       */
/*********************************************************************/
DrawInsert:
  Line.0 = 0
  Line = "INSERT INTO" TABLE "("
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = '"'SQLDA.I.SQLNAME'"'
    If Length(Line)+Length(ColName) > BndSize THEN
    Do
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
      Line = "      "
    End
    Line = Line ColName
    If I = SQLDA.SQLD Then Line = Line ')'
  End I
  If Line ^= "" Then
  Do
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
    Line = "      "
  End
```

```
  L = Line.0 + 1; Line.0 = L
  Line.L = "   VALUES ("
  L = Line.0 + 1; Line.0 = L
  Line.L = ,
  "-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE"
  Do I = 1 To SQLDA.SQLD
    If SQLDA.SQLD > 1 & I < SQLDA.SQLD Then
```

```
      Line = "                         , --"
    Else
      Line = "                         ) --"
    Line = Line Left(SQLDA.I.SQLNAME,18)
    Type  = SQLDA.I.SQLTYPE
    Null  = Type//2
    If Null Then Type = Type - 1
    Len   = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
    Scale = SQLDA.I.SQLLEN.SQLSCALE
    Select
    When (Type = CHTYPE   ,
         |Type = VCHTYPE  ,
         |Type = LVCHTYPE ,
         |Type = GRTYP    ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE  ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE   ) THEN
      Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE)")"
    Otherwise
      Type = SQLTYPES.Type
    End
    Line = Line Type
    If Null = 0 Then
    Line = Line "NOT NULL"
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
  End I
  Return
```

```
/*********************************************************************/
/* Draw UPDATE                                                       */
/*********************************************************************/
DrawUpdate:
  Line.0 = 1
  Line.1 = "UPDATE" TABLE "SET"
  L = Line.0 + 1; Line.0 = L
  Line.L = ,
  "-- COLUMN NAME           ENTER VALUES BELOW      DATA TYPE"
  Do I = 1 To SQLDA.SQLD
    If I = 1 Then
      Line = "  "
    Else
      Line = " ,"
    Line = Line Left('"'SQLDA.I.SQLNAME'"=',21)
    Line = Line Left(" ",20)
    Type  = SQLDA.I.SQLTYPE
    Null  = Type//2
    If Null Then Type = Type - 1
    Len   = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
    Scale = SQLDA.I.SQLLEN.SQLSCALE
    Select
    When (Type = CHTYPE   ,
         |Type = VCHTYPE  ,
         |Type = LVCHTYPE ,
         |Type = GRTYP    ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE  ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE   ) THEN
      Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE)")"
    Otherwise
      Type = SQLTYPES.Type
    End
    Line = Line "--" Type
    If Null = 0 Then
    Line = Line "NOT NULL"
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
  End I
  L = Line.0 + 1; Line.0 = L
  Line.L = "WHERE"
  Return
```

```
/*********************************************************************/
/* Draw LOAD                                                         */
/*********************************************************************/
DrawLoad:
  Line.0 = 1
  Line.1 = "LOAD DATA INDDN SYSREC INTO TABLE" TABLE
  Position = 1
  Do I = 1 To SQLDA.SQLD
    If I = 1 Then
      Line = " ("
    Else
      Line = " ,"
    Line = Line Left('"'SQLDA.I.SQLNAME'"',20)
    Line = Line "POSITION("RIGHT(POSITION,5)")"
    Type  = SQLDA.I.SQLTYPE
    Null  = Type//2
    If Null Then Type = Type - 1
    Len   = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
    Scale = SQLDA.I.SQLLEN.SQLSCALE
    Select
    When (Type = CHTYPE   ,
         |Type = GRTYP    ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE  ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE   ) THEN
    Do
      Type = SQLTYPES.Type "EXTERNAL"
      Type = Type"("STRIP(PRCSN)","STRIP(SCALE)")"
      Len  = (PRCSN+2)%2
    End
    When (Type = DATYPE   ,
         |Type = TITYPE   ,
         |Type = TSTYPE   ) THEN
      Type = SQLTYPES.Type "EXTERNAL"
    Otherwise
      Type = SQLTYPES.Type
    End
    If   (Type = GRTYP    ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Len = Len * 2
    If   (Type = VCHTYPE  ,
         |Type = LVCHTYPE ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Len = Len + 2
    Line = Line Type
    L = Line.0 + 1; Line.0 = L
```

```
    Line.L = Line
    If Null = 1 Then
    Do
      Line = "  "
      Line = Line Left('',20)
      Line = Line "  NULLIF("RIGHT(POSITION,5)")='?'"
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
    End
    Position = Position + Len + 1
  End I
  L = Line.0 + 1; Line.0 = L
  Line.L = " )"
  Return
/*********************************************************************/
/* Display SQLCA                                                     */
/*********************************************************************/
SQLCA:
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLSTATE="SQLSTATE"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLWARN ="SQLWARN.0",",
              || SQLWARN.1",",
              || SQLWARN.2",",
              || SQLWARN.3",",
              || SQLWARN.4",",
              || SQLWARN.5",",
              || SQLWARN.6",",
              || SQLWARN.7",",
              || SQLWARN.8",",
              || SQLWARN.9",",
```

```
              || SQLWARN.10"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRD ="SQLERRD.1",",
              || SQLERRD.2",",
              || SQLERRD.3",",
              || SQLERRD.4",",
              || SQLERRD.5",",
              || SQLERRD.6"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRP ="SQLERRP"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRMC ="SQLERRMC"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLCODE ="SQLCODE"'"
  Exit 20
```

Example of how an indicator variable is used in a REXX program

The way that you use indicator variables for input host variables in REXX programs is slightly different than the way that you use indicator variables in other languages. When you want to pass a null value to a Db2 column, in addition to putting a negative value in an indicator variable, you also need to put a valid value in the corresponding host variable.

For example, the following statements set a value in the WORKDEPT column in table EMP to null:

```
SQLSTMT="UPDATE EMP" ,
  "SET WORKDEPT = ?"
HVWORKDEPT='000'
INDWORKDEPT=-1
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING :HVWORKDEPT :INDWORKDEPT"
```

In the following program, the phone number for employee Haas is selected into variable HVPhone. After the SELECT statement executes, if no phone number for employee Haas is found, indicator variable INDPhone contains -1.

```
'SUBCOM DSNREXX'
IF RC THEN ,
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX
'CONNECT' 'DSN'
SQLSTMT = ,
  "SELECT PHONENO FROM DSN8B10.EMP WHERE LASTNAME='HAAS'"
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL PREPARE S1 FROM :SQLSTMT"
Say "SQLCODE from PREPARE is "SQLCODE
"EXECSQL OPEN C1"
Say "SQLCODE from OPEN is "SQLCODE
"EXECSQL FETCH C1 INTO :HVPhone :INDPhone"
Say "SQLCODE from FETCH is "SQLCODE
If INDPhone < 0 Then ,
Say 'Phone number for Haas is null.'
"EXECSQL CLOSE C1"
Say "SQLCODE from CLOSE is "SQLCODE
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
```

Example REXX programs for LOB data

Db2 programs in REXX can use LOB host variables and file reference variables.

Example of using simple LOB host variables in a REXX program

```
/* REXX exec to use a LOB in a host var  */

ssid = "VA1A" ;

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD","DSNREXX","DSNREXX") ;
say "rc from rxsubcom add=" rc

Address DSNREXX ;

"CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed.";
  call sqlca
```

```
  exit 8 ;
end ;

stmt = "DROP TABLE REXXCLOB" ;
Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

stmt = "CREATE TABLE REXXCLOB (" || ,
       "C1 CLOB(2M))" ;

Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/* Insert into the CLOB table */

data = "THIS IS A SHORT CLOB, BUT IT IS A CLOB" ;

stmt = "INSERT INTO REXXCLOB (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

mydata = copies('Z',75000) ;
say "length of :mydata="length(mydata) ;

Address DSNREXX "EXECSQL EXECUTE S1 USING :MYDATA" ;

say "RC/SQLCODE after EXECUTE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;
/*
var1 = copies(' ',2048000) ;
say "length of :var1="length(var1) ;
*/
stmt = "SELECT C1, LENGTH(C1) FROM REXXCLOB" ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL FETCH C1 INTO :VAR1, :VAR2" ;

say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

say "length(var1)="length(var1) ;
say "var2="var2 ;

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/**********************************************************************\
*  Disconnect from the DB2 system.                                      *
\**********************************************************************/
"DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE","DSNREXX","DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
exit 16
```

```
/*******************************************************************/
/* Error handling routine for bad SQL codes - just report and end.   */
/*******************************************************************/
SQLCA:
/*******************************************************************/
SAY "Error. SQLCODE  = >"SQLCODE"<"
SAY "       SQLSTATE = >"SQLSTATE"<"
SAY "       SQLERRMC = >"SQLERRMC"<"
SAY "       SQLERRP  = >"SQLERRP"<"
SAY "       SQLERRD.1= >"SQLERRD.1"<"
SAY "       SQLERRD.2= >"SQLERRD.2"<"
SAY "       SQLERRD.3= >"SQLERRD.3"<"
SAY "       SQLERRD.4= >"SQLERRD.4"<"
SAY "       SQLERRD.5= >"SQLERRD.5"<"
SAY "       SQLERRD.6= >"SQLERRD.6"<"
SAY "       SQLWARN.0= >"SQLWARN.0"<"
SAY "       SQLWARN.1= >"SQLWARN.1"<"
SAY "       SQLWARN.2= >"SQLWARN.2"<"
SAY "       SQLWARN.3= >"SQLWARN.3"<"
SAY "       SQLWARN.4= >"SQLWARN.4"<"
SAY "       SQLWARN.5= >"SQLWARN.5"<"
SAY "       SQLWARN.6= >"SQLWARN.6"<"
SAY "       SQLWARN.7= >"SQLWARN.7"<"
SAY "       SQLWARN.8= >"SQLWARN.8"<"
SAY "       SQLWARN.9= >"SQLWARN.9"<"
SAY "       SQLWARN.10= >"SQLWARN.10"<"
return ;
```

Example of using LOB data with an SQLDA in a REXX program

```
/* REXX EXEC TO INSERT A LOB USING SQLDA */

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD","DSNREXX","DSNREXX") ;
say "rc from rxsubcom add=" rc

ssid = "VA1A" ;
Address DSNREXX "CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed.";
  call sqlca
  exit 8 ;
end ;

stmt = "DROP TABLE REXXCLOB" ;
Address DSNREXX "EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

stmt = "CREATE TABLE REXXCLOB (" || ,
       "C1 CLOB(1M))" ;

Address DSNREXX "EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/* Insert into the CLOB table */

stmt = "INSERT INTO REXXCLOB (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 INTO :D1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

mydata = copies('A',1048560) ;   /* ~1M */

d1.sqld = 1 ;
d1.1.sqltype = 408 ;
d1.1.sqllongl= length(mydata) ;
d1.1.sqldata = mydata ;

say "length of mydata is" length(mydata) ;
Address DSNREXX "EXECSQL EXECUTE S1 USING DESCRIPTOR :D1" ;

say "RC/SQLCODE after EXECUTE S1, USING D1 is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;
```

```
stmt = "SELECT C1, LENGTH(C1) AS LENGTH FROM REXXCLOB" ;
Address DSNREXX "EXECSQL PREPARE S1 INTO :OUTDA FROM :STMT"

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

say "After PREPARE INTO, SQLDA looks like:"
say "  outda.sqld=>"outda.sqld"<" ;
do i = 1 to outda.sqld ;
  say "   "
  say "  outda."i".sqlname=>"outda.i.sqlname"<" ;
  say "  outda."i".sqltype=>"outda.i.sqltype"<" ;
end ;
say " "

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Do forever ;
  Address DSNREXX "EXECSQL FETCH C1 INTO DESCRIPTOR :OUTDA" ;

  say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
  if rc <> 0 then call sqlca ;
  if sqlcode = 100 then leave ;  /* do forever */
  say "outda.sqld=>"outda.sqld"<" ;

  do i = 1 to outda.sqld ;
     say i": sqlname =>"outda.i.sqlname"<" ;
     say i": sqltype =>"outda.i.sqltype"<" ;
     say i": sqllen  =>"outda.i.sqllen"<"  ;
     say i": sqllongl=>"outda.i.sqllongl"<"
     say i": length  =>"length(outda.i.sqldata)"<" ;
     if length(outda.i.sqldata) > 62 then
        say i": sqldata =>"substr(outda.i.sqldata,1,62)"<..." ;
     else
        say i": sqldata =>"outda.i.sqldata"<" ;
     say ' ' ;
  end ;
end ; /* do forever */

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/*********************************************************************\
*  Disconnect from the DB2 system.                                    *
\*********************************************************************/
Address DSNREXX "DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE","DSNREXX","DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
exit 16

/*********************************************************************/
/* Error handling routine for bad SQL codes - just report and end.   */
/*********************************************************************/
SQLCA:
/*********************************************************************/
SAY "       SQLCODE  = >"SQLCODE"<"
SAY "       SQLSTATE = >"SQLSTATE"<"
SAY "       SQLERRMC = >"SQLERRMC"<"
SAY "       SQLERRP  = >"SQLERRP"<"
SAY "       SQLERRD.1= >"SQLERRD.1"<"
SAY "       SQLERRD.2= >"SQLERRD.2"<"
SAY "       SQLERRD.3= >"SQLERRD.3"<"
SAY "       SQLERRD.4= >"SQLERRD.4"<"
SAY "       SQLERRD.5= >"SQLERRD.5"<"
SAY "       SQLERRD.6= >"SQLERRD.6"<"
SAY "       SQLWARN.0= >"SQLWARN.0"<"
SAY "       SQLWARN.1= >"SQLWARN.1"<"
```

```
SAY "       SQLWARN.2= >"SQLWARN.2"<"
SAY "       SQLWARN.3= >"SQLWARN.3"<"
SAY "       SQLWARN.4= >"SQLWARN.4"<"
SAY "       SQLWARN.5= >"SQLWARN.5"<"
SAY "       SQLWARN.6= >"SQLWARN.6"<"
SAY "       SQLWARN.7= >"SQLWARN.7"<"
SAY "       SQLWARN.8= >"SQLWARN.8"<"
SAY "       SQLWARN.9= >"SQLWARN.9"<"
SAY "       SQLWARN.10= >"SQLWARN.10"<"
return ;
```

Example of using LOB File Reference Variables in a REXX program

```
/* REXX EXEC TO USE A CLOB FILE REFERENCE VARIABLE */

ssid = "VA1A" ;

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD","DSNREXX","DSNREXX") ;
say "rc from rxsubcom add=" rc

Address DSNREXX ;

"CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed.";
  call sqlca_error
  exit 8 ;
end ;

stmt = "DROP TABLE REXXFRV" ;
Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 & sqlcode <> -204 then call sqlca_error ;

stmt = "CREATE TABLE REXXFRV (" || ,
       "C1 CLOB(2M))" ;

Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

/*
   Write the CLOB to the preallocated file
*/
lines = 1500; /* enough 80 byte lines to make 2,000,000 bytes */
data.1 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 01" ;
data.2 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 02" ;
data.3 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 03" ;
data.4 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 04" ;
data.5 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 05" ;
data.6 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 06" ;
data.7 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 07" ;
data.8 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 08" ;
data.9 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 09" ;
data.10= "THIS IS A SHORT CLOB, BUT IT IS A CLOB 10" ;
data.0 = 10 ;

say 'data. stem initialized' ;

Do i = 1 to data.0 ;
  data.i = left(data.i,131) ;
end ;

say 'data. stem padded to 131' ;

Do i = 1 to lines ;
  Address MVS "EXECIO" data.0 "DISKW FRVFILE (stem data." ;
  if rc <> 0 then do ;
     say 'rc from execio='rc ;
     signal bad_write ;
  end ;
end ;

/* Close the file */
```

```
Address MVS "EXECIO 0 DISKW FRVFILE (FINIS" ;

/*
   The file now has to be freed.  Otherwise, a
   SQLCODE -452, reason 12 at location 210 will be
   issued.
*/

Address TSO "FREE FI(FRVFILE)" ;
if rc <> 0 then signal bad_free ;

stmt = "INSERT INTO REXXFRV (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

/*
   Build the special SQLDA used by REXX for working with
   LOBs.
*/

mysqlda.sqld = 1 ;
mysqlda.1.sqltype = 920      /* clob file ref var */
mysqlda.1.sqlind  = 0 ;      /* not null */

/*
   Note for a file reference variable, there is
   no SQLDATA value.  Just use SQLDATA as part of the stem
   for .name and .fileoption, which are required for FRVs.
*/

mysqlda.1.sqldata.name = "SYSADM.FRV" ;  /* file name */

/*
   There are 4 fileoptions that can be set, and you can
   specify the value via text or a number.  Here are the
   allowable values:

     SQL_FILE_READ      or 2
     SQL_FILE_CREATE    or 8
     SQL_FILE_OVERWRITE or 16
     SQL_FILE_APPEND    or 32
*/

mysqlda.1.sqldata.fileoption = "SQL_FILE_READ" ;

/*
   sqllen is the length of the file name
*/
mysqlda.1.sqllen  = length(mysqlda.1.sqldata.name) ;


Address DSNREXX "EXECSQL EXECUTE S1 USING DESCRIPTOR :MYSQLDA";

say "RC/SQLCODE after EXECUTE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

stmt = "SELECT C1, LENGTH(C1) FROM REXXFRV" ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error;

Address DSNREXX "EXECSQL FETCH C1 INTO :VAR1, :VAR2" ;

say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

say "length(var1)="length(var1) ;
say "var1=" ;
```

```
say var1 ;

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/*********************************************************************\
*  Disconnect from the DB2 system.                                    *
\*********************************************************************/
"DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE","DSNREXX","DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
if sqlcode > 0 then return ;
exit 8 ;

/********************************************************************/
/* Error handling routine for bad SQL codes - just report and end.   */
/********************************************************************/
SQLCA:
/********************************************************************/
SAY "Error. SQLCODE  = >"SQLCODE"<"
SAY "       SQLSTATE = >"SQLSTATE"<"
SAY "       SQLERRMC = >"SQLERRMC"<"
SAY "       SQLERRP  = >"SQLERRP"<"
SAY "       SQLERRD.1= >"SQLERRD.1"<"
SAY "       SQLERRD.2= >"SQLERRD.2"<"
SAY "       SQLERRD.3= >"SQLERRD.3"<"
SAY "       SQLERRD.4= >"SQLERRD.4"<"
SAY "       SQLERRD.5= >"SQLERRD.5"<"
SAY "       SQLERRD.6= >"SQLERRD.6"<"
SAY "       SQLWARN.0= >"SQLWARN.0"<"
SAY "       SQLWARN.1= >"SQLWARN.1"<"
SAY "       SQLWARN.2= >"SQLWARN.2"<"
SAY "       SQLWARN.3= >"SQLWARN.3"<"
SAY "       SQLWARN.4= >"SQLWARN.4"<"
SAY "       SQLWARN.5= >"SQLWARN.5"<"
SAY "       SQLWARN.6= >"SQLWARN.6"<"
SAY "       SQLWARN.7= >"SQLWARN.7"<"
SAY "       SQLWARN.8= >"SQLWARN.8"<"
SAY "       SQLWARN.9= >"SQLWARN.9"<"
SAY "       SQLWARN.10= >"SQLWARN.10"<"
return ;
```

Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX

When Db2 prepares a REXX program that contains SQL statements, Db2 automatically includes an SQLCA in the program.

About this task

The REXX SQLCA differs from the SQLCA for other languages. The REXX SQLCA consists of a set of separate variables, rather than a structure.

The SQLCA has the following forms:

- A set of simple variables
- A set of compound variables that begin with the stem SQLCA

The simple variables is the default form of the SQLCA. Using CALL SQLEXEC results in the compound stem variables. Otherwise, the attachment command used determines the form of the SQLCA. If you use the `ADDRESS DSNREXX 'CONNECT'` *`ssid`* syntax to connect to Db2, the SQLCA variables are a set of simple variables. If you use the `CALL SQLDBS 'ATTACH TO'` syntax to connect to Db2, the SQLCA variables are compound variables that begin with the stem SQLCA.

Switching forms of the SQLCA within an application is not recommended.

Related tasks

Checking the execution of SQL statements
After executing an SQL statement, your program should check for any errors before you commit the data and handle the errors that they represent.

Checking the execution of SQL statements by using the SQLCA
One way to check whether an SQL statement executed successfully is to use the SQL communication area (SQLCA). This area is set apart for communication with Db2.

Checking the execution of SQL statements by using SQLCODE and SQLSTATE
Whenever an SQL statement executes, the SQLCODE and SQLSTATE fields of the SQLCA receive a return code.

Defining the items that your program can use to check whether an SQL statement executed successfully
If your program contains SQL statements, the program should define some infrastructure so that it can check whether the statements executed successfully. You can either include an SQL communications area (SQLCA), which contains SQLCODE and SQLSTATE variables, or declare individual SQLCODE and SQLSTATE host variables.

Defining SQL descriptor areas (SQLDA) in REXX

If your program includes certain SQL statements, you must define at least one SQL descriptor area (SQLDA). Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Procedure

Code the SQLDA declarations directly in your program.

Each SQLDA consists of a set of REXX variables with a common stem. The stem must be a REXX variable name that contains no periods and is the same as the value of *descriptor-name* that you specify when you use the SQLDA in an SQL statement.

Restrictions:

- You must place SQLDA declarations before the first SQL statement that references the data descriptor, unless you use the TWOPASS SQL processing option.
- You cannot use the SQL INCLUDE statement for the SQLDA, because it is not supported in COBOL.

Related tasks

Defining SQL descriptor areas (SQLDA)
If your program includes certain SQL statements, you must define at least one *SQL descriptor area (SQLDA)*. Depending on the context in which it is used, the SQLDA stores information about prepared SQL statements or host variables. This information can then be read by either the application program or Db2.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Equivalent SQL and REXX data types

All REXX data is string data. Therefore, when a REXX program assigns input data to a column, Db2 converts the data from a string type to the column type. When a REXX program assigns column data to an output variable, Db2 converts the data from the column type to a string type.

When you assign input data to a Db2 table column, you can either let Db2 determine the type that your input data represents, or you can use an SQLDA to tell Db2 the intended type of the input data.

When a REXX program assigns data to a column, it can either let Db2 determine the data type or use an SQLDA to specify the intended data type. If the program lets Db2 assign a data type for the input data, Db2 bases its choice on the input string format.

The following table shows the SQL data types that Db2 assigns to input data and the corresponding formats for that data. The two SQLTYPE values that are listed for each data type are the value for a column that does not accept null values and the value for a column that accepts null values.

Table 120. SQL input data types and REXX data formats

SQL data type assigned by Db2	SQLTYPE for data type	REXX input data format
INTEGER	496/497	A string of numerics that does not contain a decimal point or exponent identifier. The first character can be a plus (+) or minus (-) sign. The number that is represented must be between -2147483648 and 2147483647, inclusive.
BIGINT	492/493	A string of numbers that does not contain a decimal point or an exponent identifier. The first character can be a plus (+) or minus (-) sign. The number that is represented must be between -9223372036854775808 and -2147483648, inclusive, or between 2147483648 and 9223372036854775807.
DECIMAL(*p,s*)	484/485	One of the following formats: • A string of numerics that contains a decimal point but no exponent identifier. *p* represents the precision and *s* represents the scale of the decimal number that the string represents. The first character can be a plus (+) or minus (-) sign. • A string of numerics that does not contain a decimal point or an exponent identifier. The first character can be a plus (+) or minus (-) sign. The number that is represented is less than -9223372036854775808 or greater than 9223372036854775807.
FLOAT	480/481	A string that represents a number in scientific notation. The string consists of a series of numerics followed by an exponent identifier (an E or e followed by an optional plus (+) or minus (-) sign and a series of numerics). The string can begin with a plus (+) or minus (-) sign.
VARCHAR(*n*)	448/449	One of the following formats: • A string of length *n*, enclosed in single or double quotation marks. • The character X or x, followed by a string enclosed in single or double quotation marks. The string within the quotation marks has a length of 2**n* bytes and is the hexadecimal representation of a string of *n* characters. • A string of length *n* that does not have a numeric or graphic format, and does not satisfy either of the previous conditions.
VARGRAPHIC(*n*)	464/465	One of the following formats: • The character G, g, N, or n, followed by a string enclosed in single or double quotation marks. The string within the quotation marks begins with a shift-out character (X'0E') and ends with a shift-in character (X'0F'). Between the shift-out character and shift-in character are *n* double-byte characters. • The characters GX, Gx, gX, or gx, followed by a string enclosed in single or double quotation marks. The string within the quotation marks has a length of 4**n* bytes and is the hexadecimal representation of a string of *n* double-byte characters.

For example, when Db2 executes the following statements to update the MIDINIT column of the EMP table, Db2 must determine a data type for HVMIDINIT:

```
SQLSTMT="UPDATE EMP" ,
  "SET MIDINIT = ?"  ,
  "WHERE EMPNO = '000200'"
"EXECSQL PREPARE S100 FROM :SQLSTMT"
HVMIDINIT='H'
"EXECSQL EXECUTE S100 USING" ,
   ":HVMIDINIT"
```

Because the data that is assigned to HVMIDINIT has a format that fits a character data type, Db2 REXX Language Support assigns a VARCHAR type to the input data.

If you do not assign a value to a host variable before you assign the host variable to a column, Db2 returns an error code.

Related concepts

Compatibility of SQL and language data types
The host variable data types that are used in SQL statements must be compatible with the data types of the columns with which you intend to use them.

Accessing the Db2 REXX language support application programming interfaces

Db2 REXX Language Support includes several application programming interfaces that enable your REXX program to connect to a Db2 subsystem and execute SQL statements.

About this task

Db2 REXX Language Support includes the following application programming interfaces:

DSNREXX CONNECT

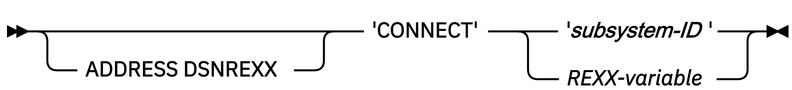
Identifies the REXX task as a connected user of the specified Db2 subsystem. The DSNREXX plan resources are allocated by establishing an allied thread.

You should not confuse the DSNREXX CONNECT command with the Db2 SQL CONNECT statement.

You must execute the DSNREXX CONNECT command before your REXX program can execute SQL statements. Do not use the DSNREXX CONNECT command from a stored procedure.

A currently connected REXX task must be disconnected before switching to a different Db2 subsystem.

The syntax of the DSNREXX CONNECT command is:



Notes:

1. CALL SQLDBS 'ATTACH TO' *ssid* is an alternative to ADDRESS DSNREXX 'CONNECT' *ssid*.
2. The *REXX-variable* or '*subsystem-ID*' string may also be a single member name in a data sharing group or the group attachment name.

The following example illustrates how to establish remote connections through the DSNREXX interface.

```
/* REXX */
/* Sample to connect to remote subsystems */
/* Connect to the local subsystem */
ADDRESS DSNREXX 'CONNECT' 'DB01'
```

```
/* Now connect to multiple remote subsystems */
ADDRESS DSNREXX 'EXECSQL CONNECT TO REMOTESYS1'
.
.
.
ADDRESS DSNREXX 'EXECSQL CONNECT TO REMOTESYS2'
.
.
.
```

DSNREXX EXECSQL

Executes SQL statements in REXX programs.

The syntax of the DSNREXX EXECSQL command is:

Notes:

1. CALL 'SQLEXEC' "SQL-statement" is an alternative to ADDRESS DSNREXX 'EXECSQL' "SQL-statement".
2. 'EXECSQL' and "SQL-statement" can be enclosed in either single or double quotation marks.

DSNREXX DISCONNECT

Deallocates the DSNREXX plan and removes the REXX task as a connected user of Db2.

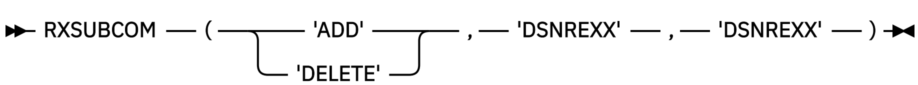
You should execute the DSNREXX DISCONNECT command to release resources that are held by Db2. Otherwise resources are not released until the REXX task terminates.

Do not use the DSNREXX DISCONNECT command from a stored procedure.

The syntax of the DSNREXX DISCONNECT command is:

Note: CALL SQLDBS 'DETACH' is an alternative to ADDRESS DSNREXX 'DISCONNECT'.

These application programming interfaces are available through the DSNREXX host command environment. To make DSNREXX available to the application, invoke the RXSUBCOM function. The syntax is:



The ADD function adds DSNREXX to the REXX host command environment table. The DELETE function deletes DSNREXX from the REXX host command environment table.

The following example illustrates REXX code that makes DSNREXX available to an application.

```
'SUBCOM DSNREXX'                        /* HOST CMD ENV AVAILABLE?     */
IF RC THEN                              /* IF NOT, MAKE IT AVAILABLE   */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
                                        /* ADD HOST CMD ENVIRONMENT    */
ADDRESS DSNREXX                         /* SEND ALL COMMANDS OTHER     */
                                        /* THAN REXX INSTRUCTIONS TO   */
                                        /* DSNREXX                     */
```

```
                                        /* CALL CONNECT, EXECSQL, AND  */
                                        /* DISCONNECT INTERFACES       */
⋮
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
                                        /* WHEN DONE WITH              */
                                        /* DSNREXX, REMOVE IT.         */
```

Related concepts

REXX stored procedures
A REXX stored procedure is similar to any other REXX procedure and follows the same rules as stored procedures in other languages. A REXX stored procedure receives input parameters, executes REXX commands, optionally executes SQL statements, and returns at most one output parameter. However, a few differences exist.

Ensuring that Db2 correctly interprets character input data in REXX programs

Db2 REXX Language Support might incorrectly interpret character literals as graphic or numeric literals unless you mark them correctly.

Procedure

Precede and follow character literals with a double quotation mark, followed by a single quotation mark, followed by another double quotation mark (`"'"`).
For example, Specify the string the string 100 as `"'"100"'"`.

Enclosing the string in apostrophes is not adequate, because REXX removes the apostrophes when it assigns a literal to a variable. For example, suppose that you want to pass the value in a host variable called stringvar to Db2. The value that you want to pass is the string '100'. First, you assign the string to the host variable by issuing the following REXX command:

```
stringvar = '100'
```

After the command executes, stringvar contains the characters 100 (without the apostrophes). Db2 REXX Language Support then passes the numeric value 100 to Db2, which is not what you intended.

However, suppose that you write the following command:

```
stringvar = "'"100"'"
```

In this case, REXX assigns the string '100' to stringvar, including the single quotation marks. Db2 REXX Language Support then passes the string '100' to Db2, which is the result that you want.

Passing the data type of an input data type to Db2 for REXX programs

In certain situations, you should tell Db2 the data type to use for input data in a REXX program. For example, if you are assigning or comparing input data to columns of type SMALLINT, CHAR, or GRAPHIC, you should tell Db2 to use those data types.

About this task

Db2 does not assign data types of SMALLINT, CHAR, or GRAPHIC to input data. If you assign or compare this data to columns of type SMALLINT, CHAR, or GRAPHIC, Db2 must do more work than if the data types of the input data and columns match.

Procedure

Use an SQLDA.

Examples

Example: Specifying CHAR as an input data type

Suppose that you want to tell Db2 that the data with which you update the MIDINIT column of the EMP table is of type CHAR, rather than VARCHAR. You need to set up an SQLDA that contains a description of a CHAR column, and then prepare and execute the UPDATE statement using that SQLDA, as shown in the following example.

```
INSQLDA.SQLD = 1                /* SQLDA contains one variable   */
INSQLDA.1.SQLTYPE = 453         /* Type of the variable is CHAR, */
                                /* and the value can be null     */
INSQLDA.1.SQLLEN  = 1           /* Length of the variable is 1   */
INSQLDA.1.SQLDATA = 'H'         /* Value in variable is H        */
INSQLDA.1.SQLIND  = 0           /* Input variable is not null    */
SQLSTMT="UPDATE EMP" ,
  "SET MIDINIT = ?"  ,
  "WHERE EMPNO = '000200'"
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING DESCRIPTOR :INSQLDA"
```

Example: specifying the input data type as DECIMAL with precision and scale

Suppose that you want to tell Db2 that the data is of type DECIMAL with precision and nonzero scale. You need to set up an SQLDA that contains a description of a DECIMAL column, as shown in the following example.

```
INSQLDA.SQLD = 1                         /* SQLDA contains one variable  */
INSQLDA.1.SQLTYPE = 484                  /* Type of variable is DECIMAL  */
INSQLDA.1.SQLLEN.SQLPRECISION = 18       /* Precision of variable is 18  */
INSQLDA.1.SQLLEN.SQLSCALE = 8            /* Scale of variable is 8       */
INSQLDA.1.SQLDATA = 9876543210.87654321  /* Value in variable            */
```

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

The REXX SQLDA (Db2 SQL)

Setting the isolation level of SQL statements in a REXX program

Isolation levels specify the locking behavior for SQL statements. You can set the isolation level for SQL statements in your REXX program to repeatable read (RR), read stability (RS), cursor stability (CS), or uncommitted read (UR).

Procedure

Execute the SET CURRENT PACKAGESET statement to select one of the following Db2 REXX Language Support packages with the isolation level that you need.

Table 121. Db2 REXX Language Support packages and associated isolation levels

Package name[a]	Isolation level
DSNREXRR	Repeatable read (RR)
DSNREXRS	Read stability (RS)
DSNREXCS	Cursor stability (CS)
DSNREXUR	Uncommitted read (UR)

Note:

a. These packages enable your program to access Db2 and are bound when you install Db2 REXX Language Support.

For example, to change the isolation level to cursor stability, execute the following SQL statement:

```
"EXECSQL SET CURRENT PACKAGESET='DSNREXCS'"
```

Retrieving data from Db2 tables in REXX programs

All output data in REXX programs is string data. Although, you can determine the data type that the data represents from its format and from the data type of the column from which the data was retrieved.

About this task

The following table gives the format for each type of output data.

Table 122. SQL output data types and REXX data formats

SQL data type	**REXX output data format**
SMALLINT INTEGER BIGINT	A string of numerics that does not contain leading zeroes, a decimal point, or an exponent identifier. If the string represents a negative number, it begins with a minus (-) sign. The numeric value is between -9223372036854775808 and 9223372036854775807, inclusive.
DECIMAL(*p,s*)	A string of numerics with one of the following formats: • Contains a decimal point but not an exponent identifier. The string is padded with zeroes to match the scale of the corresponding table column. If the value represents a negative number, it begins with a minus (-) sign. • Does not contain a decimal point or an exponent identifier. The numeric value is less than -9223372036854775808 or greater than 9223372036854775807. If the value is negative, it begins with a minus (-) sign.
FLOAT(*n*) REAL DOUBLE	A string that represents a number in scientific notation. The string consists of a numeric, a decimal point, a series of numerics, and an exponent identifier. The exponent identifier is an E followed by a minus (-) sign and a series of numerics if the number is between -1 and 1. Otherwise, the exponent identifier is an E followed by a series of numerics. If the string represents a negative number, it begins with a minus (-) sign.
DECFLOAT	REXX emulates the DECFLOAT data type with DOUBLE, so support for DECFLOAT is limited to the REXX support for DOUBLE. The following special values are not supported: • INFINITY • SNAN • NAN
CHAR(*n*) VARCHAR(*n*) CLOB(*n*) BLOB(*n*)	A character string or LOB value of length *n* bytes. The string is not enclosed in single or double quotation marks.
GRAPHIC(*n*) VARGRAPHIC(*n*) DBCLOB(*n*)	A string of length 2**n* bytes. Each pair of bytes represents a double-byte character. This string does not contain a leading G, is not enclosed in quotation marks, and does not contain shift-out or shift-in characters.

Because you cannot use the SELECT INTO statement in a REXX procedure, to retrieve data from a Db2 table you must prepare a SELECT statement, open a cursor for the prepared statement, and then fetch rows into host variables or an SQLDA using the cursor. The following example demonstrates how you can retrieve data from a Db2 table using an SQLDA:

```
SQLSTMT= ,
'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,' ,
'  WORKDEPT, PHONENO, HIREDATE, JOB,' ,
'  EDLEVEL, SEX, BIRTHDATE, SALARY,' ,
'  BONUS, COMM' ,
'  FROM EMP'
EXECSQL DECLARE C1 CURSOR FOR S1
EXECSQL PREPARE S1 INTO :OUTSQLDA FROM :SQLSTMT
EXECSQL OPEN C1
Do Until(SQLCODE ¬= 0)
  EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA
  If SQLCODE = 0 Then Do
    Line = ''
    Do I = 1 To OUTSQLDA.SQLD
      Line = Line OUTSQLDA.I.SQLDATA
    End I
    Say Line
   End
End
```

Cursors and statement names in REXX

In REXX applications that contain SQL statements, you must use a predefined set of names for cursors or prepared statements.

The following names are valid for cursors and prepared statements in REXX applications:

c1 to c100
Cursor names for DECLARE CURSOR, OPEN, CLOSE, and FETCH statements. By default, c1 to c100 are defined with the WITH RETURN clause, and c51 to c100 are defined with the WITH HOLD clause. You can use the ATTRIBUTES clause of the PREPARE statement to override these attributes or add additional attributes. For example, you might want to add attributes to make your cursor scrollable.

c101 to c200
Cursor names for ALLOCATE, DESCRIBE, FETCH, and CLOSE statements that are used to retrieve result sets in a program that calls a stored procedure.

s1 to s100
Prepared statement names for DECLARE STATEMENT, PREPARE, DESCRIBE, and EXECUTE statements.

Use only the predefined names for cursors and statements. When you associate a cursor name with a statement name in a DECLARE CURSOR statement, the cursor name and the statement must have the same number. For example, if you declare cursor c1, you need to declare it for statement s1:

```
EXECSQL 'DECLARE C1 CURSOR FOR S1'
```

Do not use any of the predefined names as host variables names.

Handling SQL error codes in REXX applications

REXX applications can request more information about SQL error codes by using the DSNTIAR subroutine or issuing a GET DIAGNOSTICS statement.

Procedure

Db2 does not support the SQL WHENEVER statement in a REXX program. To handle SQL errors and warnings, use the following methods:

- To test for SQL errors or warnings, test the SQLCODE or SQLSTATE value and the SQLWARN. values after each EXECSQL call. This method does not detect errors in the REXX interface to Db2.
- To test for SQL errors or warnings or errors or warnings from the REXX interface to Db2, test the REXX RC variable after each EXECSQL call.

  The following table lists the values of the RC variable.

Table 123. REXX return codes after SQL statements

Return code	Meaning
0	No SQL warning or error occurred.
+1	An SQL warning occurred.
-1	An SQL error occurred.
-3	The first token after ADDRESS DSNREXX is in error. For a description of the tokens allowed, see "Accessing the Db2 REXX language support application programming interfaces" on page 745.

You can also use the REXX SIGNAL ON ERROR and SIGNAL ON FAILURE keyword instructions to detect negative values of the RC variable and transfer control to an error routine.

Related tasks

Handling SQL error codes
Application programs can request more information about SQL error codes from Db2.

Related reference

GET DIAGNOSTICS (Db2 SQL)

Chapter 5. Calling a stored procedure from your application

To run a stored procedure, you can either call it from a client program or invoke it from the command line processor.

Before you begin

Before you call a stored procedure, ensure that you have all of the following authorizations that are required to run the stored procedure:

- Authorization to execute the stored procedure that is referenced in the CALL statement.

  The authorizations that you need depend on whether the form of the CALL statement is CALL *procedure-name* or CALL :*host-variable*.
- Authorization to execute any triggers or user-defined functions that the stored procedure invokes.
- Authorization to execute the stored procedure package and any packages under the stored procedure package.

  For example, if the stored procedure invokes any user-defined functions, you need authorization to execute the packages for those user-defined functions.

About this task

An application program that calls a stored procedure can perform one or more of the following actions:

- Call more than one stored procedure.
- Call a single stored procedure more than once at the same or at different levels of nesting. However, do not assume that the variables for the stored procedures persist between calls.

  If a stored procedure runs as a main program, before each call, Language Environment reinitializes the storage that is used by the stored procedure. Program variables for the stored procedure do not persist between calls.

  If a stored procedure runs as a subprogram, Language Environment does not initialize the storage between calls. Program variables for the stored procedure can persist between calls. However, you should not assume that your program variables are available from one stored procedure call to another call for the following reasons:

  - Stored procedures from other users can run in an instance of Language Environment between two executions of your stored procedure.
  - Consecutive executions of a stored procedure might run in different stored procedure address spaces.
  - The z/OS operator might refresh Language Environment between two executions of your stored procedure.
- Call a local or remote stored procedure.

  If both the client and server application environments support two-phase commit, the coordinator controls updates between the application, the server, and the stored procedures. If either side does not support two-phase commit, updates fail.
- Mix CALL statements with other SQL statements.
- Use any of the Db2 attachment facilities.

Db2 runs stored procedures under the Db2 thread of the calling application, which means that the stored procedures are part of the caller's unit of work.

JDBC and ODBC applications: These instructions do not apply to JDBC and ODBC applications. Instead, see the following information for how to call stored procedures from those applications:

- For ODBC applications, see Stored procedure calls in a Db2 ODBC application (Db2 Programming for ODBC).
- For JDBC applications, see Calling stored procedures in JDBC applications (Db2 Application Programming for Java)

Procedure

To call a stored procedure from your application:

1. Assign values to the IN and INOUT parameters.
2. Optional: To improve application performance, initialize the length of LOB output parameters to zero.
3. If the stored procedure exists at a remote location, perform the following actions:
   a) Assign values to the OUT parameters.

      When you call a stored procedure at a remote location, the local Db2 server cannot determine whether the parameters are input (IN) or output (OUT or INOUT) parameters. Therefore, you must initialize the values of all output parameters before you call a stored procedure at a remote location.

   b) Optional: Issue an explicit CONNECT statement to connect to the remote server.

      If you do not issue this statement explicitly, you can implicitly connect to the server by using a three-part name to identify the stored procedure in the next step.

      The advantage of issuing an explicit CONNECT statement is that your CALL statement, which is described in the next step, is portable to other operating systems. The advantage of implicitly connecting is that you do not need to issue this extra CONNECT statement.

      Requirement: When deciding whether to implicitly or explicitly connect to the remote server, consider the requirement for programs that execute the ASSOCIATE LOCATORS or DESCRIBE PROCEDURE statements. You must use the same form of the procedure name on the CALL statement and on the ASSOCIATE LOCATORS or DESCRIBE PROCEDURE statement.
4. Invoke the stored procedure with the SQL CALL statement. Make sure that you pass parameter data types that are compatible.

   If the stored procedure exists on a remote server and you did not issue an explicit CONNECT statement, specify a three-part name to identify the stored procedure, and implicitly connect to the server where the stored procedure is located.

   For native SQL procedures, the active version of the stored procedure is invoked by default. Optionally, you can specify a version of the stored procedure other than the active version.

   To allow null values for parameters, use indicator variables.
5. Optional: Retrieve the status of the procedure.
6. Process any output, including the OUT and INOUT parameters.
7. If the stored procedure returns multiple result sets, retrieve those result sets.

   Recommendation: Close the result sets after you retrieve them, and issue frequent commits to prevent Db2 storage shortages and EDM POOL FULL conditions.
8. For PL/I applications, also perform the following actions:
   a) Include the run time option NOEXECOPS in your source code.
   b) Specify the compile-time option SYSTEM(MVS).

   These additional steps ensure that the linkage conventions work correctly on z/OS.
9. For C applications, include the following line in your source code:

```
#pragma runopts(PLIST(OS))
```

This code ensures that the linkage conventions work correctly on z/OS.

This option is not applicable to other operating systems. If you plan to use a C stored procedure on other platforms besides z/OS, use one of the forms of conditional compilation, as shown in the following example, to include this option only when you compile on z/OS.

Form 1

```
#ifdef MVS
 #pragma runopts(PLIST(OS))
#endif
```

Form 2

```
#ifndef WKSTN
 #pragma runopts(PLIST(OS))
#endif
```

10. Prepare the application as you would any other application by precompiling, compiling, and link-editing the application and binding the DBRM.

    If the application calls a remote stored procedure, perform the following additional steps when you bind the DBRM:

    - Bind the DBRM into a package at the local Db2 server. Use the bind option DBPROTOCOL(DRDA). If the stored procedure name cannot be resolved until run time, also specify the bind option VALIDATE(RUN). The stored procedure name might not be resolved at run time if you use a variable for the stored procedure name or if the stored procedure exists on a remote server.
    - Bind the DBRM into a package at the remote Db2 server. If your client program accesses multiple servers, bind the program at each server.
    - Bind all packages into a plan at the local Db2 server. Use the bind option DBPROTOCOL(DRDA).
11. Ensure that stored procedure completed successfully.

    If a stored procedure abnormally terminates, Db2 performs the following actions:

    - The calling program receives an SQL error as notification that the stored procedure failed.
    - Db2 places the calling program's unit of work in a must-rollback state.
    - Db2 stops the stored procedure, and subsequent calls fail, in either of the following conditions:
      - The number of abnormal terminations equals the STOP AFTER *n* FAILURES value for the stored procedure.
      - The number of abnormal terminations equals the default MAX ABEND COUNT value for the subsystem.
    - The stored procedure does not handle the abend condition, and Db2 refreshes the environment for Language Environment to recover the storage that the application uses. In most cases, the environment does not need to restart.
    - A data set is allocated in the DD statement CEEDUMP in the JCL procedure that starts the stored procedures address space. In this case, Language Environment writes a small diagnostic dump to this data set. Use the information in the dump to debug the stored procedure.
    - In a data sharing environment, the stored procedure is placed in STOPABN status only on the member where the abends occurred. A calling program can invoke the stored procedure from other members of the data sharing group. The status on all other members is STARTED.

Examples

Example 1: Simple CALL statement

The following example shows a simple CALL statement that you might use to invoke stored procedure A:

```
EXEC SQL CALL A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE, :CODE);
```

In this example, :EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE, and :CODE are host variables that you have declared earlier in your application program.

Example 2: Using a host structure for multiple parameter values

Instead of passing each parameter separately, as shown in the example of a simple CALL statement, you can pass them together as a host structure. For example, assume that you defined the following host structure in your application:

```
struct {
  char EMP[7];
  char PRJ[7];
  short ACT;
  short EMT;
  char  EMS[11];
  char  EME[11];
} empstruc;
```

You can then issue the following CALL statement to invoke stored procedure A:

```
EXEC SQL CALL A (:empstruc, :TYPE, :CODE);
```

Example 3: Calling a remote stored procedure

- The following example shows how to explicitly connect to LOCA and then issue a CALL statement:

```
EXEC SQL CONNECT TO LOCA;
EXEC SQL CALL SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
  :TYPE, :CODE);
```

- The following example shows how to implicitly connect to LOCA by specifying the three-part name for stored procedure A in the CALL statement:

```
EXEC SQL CALL LOCA.SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS,
  :EME, :TYPE, :CODE);
```

Example 4: Passing parameters that can have null values

The preceding examples assume that none of the input parameters can have null values. The following example shows how to allow for null values for the parameters by passing indicator variables in the parameter list:

```
EXEC SQL CALL A (:EMP :IEMP, :PRJ :IPRJ, :ACT :IACT,
                 :EMT :IEMT, :EMS :IEMS, :EME :IEME,
                 :TYPE :ITYPE, :CODE :ICODE);
```

In this example, :IEMP, :IPRJ, :IACT, :IEMT, :IEMS, :IEME, :ITYPE, and :ICODE are indicator variables for the parameters.

Example 5: Passing string constants and null values

The following example CALL statement passes integer and character string constants, a null value, and several host variables:

```
EXEC SQL CALL A ('000130', 'IF1000', 90, 1.0, NULL, '2009-10-01',
                 :TYPE, :CODE);
```

Example 6: of using a host variable for the stored procedure name

The following example CALL statement uses a host variable for the name of the stored procedure:

```
EXEC SQL CALL :procnm (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
  :TYPE, :CODE);
```

Assume that the stored procedure name is A. The host variable *procnm* is a character variable of length 255 or less that contains the value 'A'. Use this technique if you do not know in advance the name of the stored procedure, but you do know the parameter list convention.

Example 7: Using an SQLDA to pass parameters in a single structure

The following example CALL statement shows how to pass parameters in a single structure, the SQLDA, rather than as separate host variables:

```
EXEC SQL CALL A USING DESCRIPTOR :sqlda;
```

sqlda is the name of an SQLDA.

One advantage of using an SQLDA is that you can change the encoding scheme of the stored procedure parameter values. For example, if the subsystem on which the stored procedure runs has an EBCDIC encoding scheme, and you want to retrieve data in ASCII CCSID 437, you can specify the CCSIDs for the output parameters in the SQLVAR fields of the SQLDA.

This technique for overriding the CCSIDs of parameters is the same as the technique for overriding the CCSIDs of variables. This technique involves including dynamic SQL for varying-list SELECT statements in your program. When you use this technique, the defined encoding scheme of the parameter must be different from the encoding scheme that you specify in the SQLDA. Otherwise, no conversion occurs.

The defined encoding scheme for the parameter is the encoding scheme that you specify in the CREATE PROCEDURE statement. If you do not specify an encoding scheme in this statement, the defined encoding scheme for the parameter is the default encoding scheme for the subsystem.

Example 8: Reusable CALL statement

Because the following example CALL statement uses a host variable name for the stored procedure and an SQLDA for the parameter list, it can be reused to call different stored procedures with different parameter lists:

```
EXEC SQL CALL :procnm USING DESCRIPTOR :sqlda;
```

Your client program must assign a stored procedure name to the host variable *procnm* and load the SQLDA with the parameter information before issuing the SQL CALL statement.

Related concepts

Stored procedure parameters
You can pass information between a stored procedure and the calling application program by using parameters. Applications pass the required parameters in the SQL CALL statement. Optionally, the application can also include an indicator variable with each parameter to allow for null values or to pass large output parameter values.

Related tasks

Including dynamic SQL for varying-list SELECT statements in your program
A varying-list SELECT statement returns rows that contain an unknown number of values of unknown type. When you use this type of statement, you do not know in advance exactly what kinds of host variables you need to declare for storing the results.

Preparing an application to run on Db2 for z/OS
To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

Managing authorization for stored procedures (Managing Security)

Temporarily overriding the active version of a native SQL procedure
If you want a particular call to a native SQL procedure to use a version other than the active version, you can temporarily override the active version. Such an override might be helpful when you are testing a new version of a native SQL procedure.

Related reference

Statements (Db2 SQL)

Procedures that are supplied with Db2 (Db2 SQL)

Passing large output parameters to stored procedures by using indicator variables

If any output parameters occupy a large amount of storage, passing the entire storage area to a stored procedure can degrade performance.

About this task

In the calling program, you can specify indicator variables for large output parameters to pass only a 2-byte area to the stored procedure, but receive the entire output data area from the stored procedure. When Db2 processes the CALL statement, it inspects the parameters before moving any data. If an output parameter has a NULL indicator, Db2 determines that it does not need to copy the associated data area to the Db2 address space, which avoids the need for acquisition of extra buffers or cross-memory moves.

You can use the following procedure regardless of whether the linkage convention for the stored procedure is GENERAL, GENERAL WITH NULLS, or SQL.

Procedure

To pass large output parameters to stored procedures by using indicator variables:

1. Declare an indicator variable for every large output parameter in the stored procedure.

   If you are using the GENERAL WITH NULLS or SQL linkage convention, you must declare indicator variables for all of your parameters. In this case, you do not need to declare another indicator variable.
2. Assign a negative value to each indicator variable that is associated with a large output variable.
3. Include the indicator variables in the CALL statement.

Example

For example, suppose that a stored procedure that is defined with the GENERAL linkage convention takes one integer input parameter and one character output parameter of length 6000. You do not want to pass the 6000 byte storage area to the stored procedure. The following example PL/I program passes only 2 bytes to the stored procedure for the output variable and receives all 6000 bytes from the stored procedure:

```
DCL INTVAR BIN FIXED(31);        /* This is the input variable    */
DCL BIGVAR(6000);                /* This is the output variable   */
DCL I1 BIN FIXED(15);            /* This is an indicator variable */
⋮
I1 = -1;                         /* Setting I1 to -1 causes only  */
                                 /*  a two byte area representing */
                                 /*  I1 to be passed to the       */
                                 /*  stored procedure, instead of */
                                 /*  the 6000 byte area for BIGVAR*/
EXEC SQL CALL PROCX(:INTVAR, :BIGVAR INDICATOR :I1);
```

Related reference

Linkage conventions for external stored procedures
The linkage convention for a stored procedure can be either GENERAL, GENERAL WITH NULLS, or SQL. These linkage conventions apply to only external stored procedures.

Data types for calling stored procedures

The data types that are available for calling applications are the same as the data types that are used when retrieving or updating stored procedures.

The format of the parameters that you pass in the CALL statement in an application must be compatible with the data types of the parameters in the CREATE PROCEDURE statement.

For languages other than REXX

For all data types except LOBs, ROWIDs, locators, and VARCHARs (for C language), see the tables listed in the following table for the host data types that are compatible with the data types in the stored procedure definition.

Table 124. Listing of tables of compatible data types

Language	Compatible data types table
Assembler	"Equivalent SQL and assembler data types" on page 564
C	"Equivalent SQL and C data types" on page 611
COBOL	"Equivalent SQL and COBOL data types" on page 677
PL/I	"Equivalent SQL and PL/I data types" on page 720

Calling a stored procedure from a REXX procedure

The format of the parameters that you pass in the CALL statement in a REXX procedure must be compatible with the data types of the parameters in the CREATE PROCEDURE statement.

The following table lists each SQL data type that you can specify for the parameters in the CREATE PROCEDURE statement and the corresponding format for a REXX parameter that represents that data type.

Table 125. Parameter formats for a CALL statement in a REXX procedure

SQL data type	REXX format
SMALLINT INTEGER BIGINT	A string of numerics that does not contain a decimal point or exponent identifier. The first character can be a plus or minus sign. This format also applies to indicator variables that are passed as parameters.
DECIMAL(*p*,*s*) NUMERIC(*p*,*s*)	A string of numerics that has a decimal point but no exponent identifier. The first character can be a plus or minus sign.
REAL FLOAT(*n*) DOUBLE DECFLOAT	A string that represents a number in scientific notation. The string consists of a series of numerics followed by an exponent identifier (an E or e followed by an optional plus or minus sign and a series of numerics).
CHARACTER(*n*) VARCHAR(*n*) VARCHAR(*n*) FOR BIT DATA	A string of length *n*, enclosed in single quotation marks.
GRAPHIC(*n*) VARGRAPHIC(*n*)	The character G followed by a string enclosed in single quotation marks. The string within the quotation marks begins with a shift-out character (X'0E') and ends with a shift-in character (X'0F'). Between the shift-out character and shift-in character are *n* double-byte characters.
BINARY VARBINARY	**Recommendation:** Pass BINARY and VARBINARY values by using the SQLDA. If you specify an SQLDA when you call the stored procedure, set the SQLTYPE in the SQLDA. SQLDATA is a string of characters. If you use host variables, the REXX format of BINARY and VARBINARY data is BX followed by a string that is enclosed in a single quotation mark.

Table 125. Parameter formats for a CALL statement in a REXX procedure (continued)

SQL data type	REXX format
DATE	A string of length 10, enclosed in single quotation marks. The format of the string depends on the value of field DATE FORMAT that you specify when you install Db2.
TIME	A string of length 8, enclosed in single quotation marks. The format of the string depends on the value of field TIME FORMAT that you specify when you install Db2.
TIMESTAMP	A string of length 19 to 32, enclosed in single quotation marks. The string has the format *yyyy-mm-dd-hh.mm.ss* or *yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn*, where the number of fractional second digits can range 0 - 12.
TIMESTAMP WITH TIME ZONE	A string of length 148 to 161, enclosed in single quotation marks. The string has the format *yyyymm- dd-hh.mm.ss.nnnnnnnnnnnn±th:tm or yyyymm- dd-hh.mm.ss.nnnnnnnnnnnn ±th:tm*, where the number of fractional second digits can range 0 - 12
XML	No equivalent.

The following figure demonstrates how a REXX procedure calls the stored procedure in "REXX stored procedures" on page 287. The REXX procedure performs the following actions:

- Connects to the Db2 subsystem that was specified by the REXX procedure invoker.
- Calls the stored procedure to execute a Db2 command that was specified by the REXX procedure invoker.
- Retrieves rows from a result set that contains the command output messages.

```
/* REXX */
PARSE ARG SSID COMMAND                  /* Get the SSID to connect to */
                                        /* and the DB2 command to be  */
                                        /* executed                   */
  /****************************************************************/
  /*  Set up the host command environment for SQL calls.          */
  /****************************************************************/
"SUBCOM DSNREXX"                        /* Host cmd env available? */
IF RC THEN                              /* No--make one               */
   S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
  /****************************************************************/
  /*  Connect to the DB2 subsystem.                               */
  /****************************************************************/
ADDRESS DSNREXX "CONNECT" SSID
IF SQLCODE ¬= 0 THEN CALL SQLCA
PROC = 'COMMAND'
RESULTSIZE = 32703
RESULT = LEFT(' ',RESULTSIZE,' ')
  /****************************************************************/
  /*  Call the stored procedure that executes the DB2 command.    */
  /*  The input variable (COMMAND) contains the DB2 command.      */
  /*  The output variable (RESULT) will contain the return area   */
  /*  from the IFI COMMAND call after the stored procedure        */
  /*  executes.                                                   */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL" ,
"CALL" PROC "(:COMMAND, :RESULT)"
IF SQLCODE < 0 THEN CALL SQLCA
SAY 'RETCODE ='RETCODE
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
```

```
             || SQLWARN.2',',
             || SQLWARN.3',',
             || SQLWARN.4',',
             || SQLWARN.5',',
             || SQLWARN.6',',
             || SQLWARN.7',',
             || SQLWARN.8',',
             || SQLWARN.9',',
             || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY C2X(RESULT) "'"||RESULT||"'"
```

```
  /****************************************************************/
  /*  Display the IFI return area in hexadecimal.                 */
  /****************************************************************/
OFFSET = 4+1
TOTLEN = LENGTH(RESULT)
DO WHILE ( OFFSET <  TOTLEN )
  LEN = C2D(SUBSTR(RESULT,OFFSET,2))
  SAY SUBSTR(RESULT,OFFSET+4,LEN-4-1)
  OFFSET = OFFSET + LEN
END
  /****************************************************************/
  /*  Get information about result sets returned by the           */
  /*  stored procedure.                                           */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL DESCRIBE PROCEDURE :PROC INTO :SQLDA"
IF SQLCODE ¬= 0 THEN CALL SQLCA
DO I = 1 TO SQLDA.SQLD
  SAY "SQLDA."I".SQLNAME    ="SQLDA.I.SQLNAME";"
  SAY "SQLDA."I".SQLTYPE    ="SQLDA.I.SQLTYPE";"
  SAY "SQLDA."I".SQLLOCATOR ="SQLDA.I.SQLLOCATOR";"
END I
  /****************************************************************/
  /*  Set up a cursor to retrieve the rows from the result        */
  /*  set.                                                        */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL ASSOCIATE LOCATOR (:RESULT) WITH PROCEDURE :PROC"
IF SQLCODE ¬= 0 THEN CALL SQLCA
SAY RESULT
ADDRESS DSNREXX "EXECSQL ALLOCATE C101 CURSOR FOR RESULT SET :RESULT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
CURSOR = 'C101'
ADDRESS DSNREXX "EXECSQL DESCRIBE CURSOR :CURSOR INTO :SQLDA"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Retrieve and display the rows from the result set, which    */
  /*  contain the command output message text.                    */
  /****************************************************************/
DO UNTIL(SQLCODE ¬= 0)
  ADDRESS DSNREXX "EXECSQL FETCH C101 INTO :SEQNO, :TEXT"
  IF SQLCODE = 0 THEN
    DO
      SAY TEXT
    END
END
IF SQLCODE ¬= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL CLOSE C101"
IF SQLCODE ¬= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
```

```
  /****************************************************************/
  /*  Disconnect from the DB2 subsystem.                          */
  /****************************************************************/
ADDRESS DSNREXX "DISCONNECT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Delete the host command environment for SQL.                */
  /****************************************************************/
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV  */
RETURN
  /****************************************************************/
  /*  Routine to display the SQLCA                                */
  /****************************************************************/
SQLCA:
TRACE O
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
```

```
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
EXIT
```

Related concepts

REXX stored procedures
A REXX stored procedure is similar to any other REXX procedure and follows the same rules as stored procedures in other languages. A REXX stored procedure receives input parameters, executes REXX commands, optionally executes SQL statements, and returns at most one output parameter. However, a few differences exist.

Preparing a client program that calls a remote stored procedure

If you call a remote stored procedure from an embedded SQL application, you need to do a few extra steps when you prepare the client program. You do not need to do any extra steps when you prepare the stored procedure.

Before you begin

For an ODBC or CLI application, ensure that the Db2 packages and plan that are associated with the ODBC driver are bound to Db2. These packages and plan must be bound before you can run your application.

Procedure

To prepare a client program that calls a remote stored procedure:

1. Precompile, compile, and link-edit the client program on the local Db2 subsystem.
2. Bind the resulting DBRM into a package at the local Db2 subsystem by using the BIND PACKAGE command with the option DBPROTOCOL(DRDA).
3. Bind the same DBRM, the one for the client program, into a package at the remote location by using the BIND PACKAGE command and specifying a location name. If your client program needs to access multiple servers, bind the program at each server.
   For example, suppose that you want a client program to call a stored procedure at location LOCA. You precompile the program to produce DBRM A. Then you can use the following command to bind DBRM A into package collection COLLA at location LOCA:

   ```
   BIND PACKAGE (LOCA.COLLA) MEMBER(A)
   ```

4. Bind all packages into a plan on the local Db2 subsystem. Specify the bind option DBPROTOCOL(DRDA).
5. Bind any stored procedures that run under Db2 ODBC on a remote Db2 database server as a package at the remote site.

   Those procedures do not need to be bound into the Db2 ODBC plan.

Related tasks

Binding DBRMs to create packages (Db2 Programming for ODBC)

Related reference

BIND PACKAGE (DSN) (Db2 Commands)

How Db2 determines which stored procedure to run

A procedure is uniquely identified by its name and its qualifying schema name. You can tell Db2 exactly which stored procedure to run by qualifying it with its schema name when you call it. Otherwise, Db2 determines which stored procedure to run.

However, if you do not qualify the stored procedure name, Db2 uses the following method to determine which stored procedure to run:

1. Db2 searches the list of schema names from the PATH bind option or the CURRENT PATH special register from left to right until it finds a schema name for which a stored procedure definition exists with the name in the CALL statement.

   Db2 uses schema names from the PATH bind option for CALL statements of the following form:

   ```
   CALL procedure-name
   ```

   Db2 uses schema names from the CURRENT PATH special register for CALL statements of the following form:

   ```
   CALL host-variable
   ```

2. When Db2 finds a stored procedure definition, Db2 executes that stored procedure if the following conditions are true:

   - The caller is authorized to execute the stored procedure.
   - The stored procedure has the same number of parameters as in the CALL statement.

   If both conditions are not true, Db2 continues to go through the list of schemas until it finds a stored procedure that meets both conditions or reaches the end of the list.

3. If Db2 cannot find a suitable stored procedure, it returns an SQL error code for the CALL statement.

Calling different versions of a stored procedure from a single application

You can call different versions of a stored procedure from the same application program, even though those versions all have the same load module name.

Procedure

To call different versions of a stored procedure from a single application:

1. When you define each version of the stored procedure, use the same stored procedure name but different schema names, different COLLID values, and different WLM environments.
2. In the program that invokes the stored procedure, specify the unqualified stored procedure name in the CALL statement.
3. Use the SQL path to indicate which version of the stored procedure that the client program should call. You can choose the SQL path in several ways:

   - If the client program is not an ODBC or JDBC application, use one of the following methods:
     - Use the CALL *procedure-name* form of the CALL statement. When you bind plans or packages for the program that calls the stored procedure, bind one plan or package for each version of the stored procedure that you want to call. In the PATH bind option for each plan or package, specify the schema name of the stored procedure that you want to call.
     - Use the CALL *host-variable* form of the CALL statement. In the client program, use the SET PATH statement to specify the schema name of the stored procedure that you want to call.
   - If the client program is an ODBC or JDBC application, choose one of the following methods:
     - Use the SET PATH statement to specify the schema name of the stored procedure that you want to call.

- When you bind the stored procedure packages, specify a different collection for each stored procedure package. Use the COLLID value that you specified when defining the stored procedure to Db2.

4. When you run the client program, specify the plan or package with the PATH value that matches the schema name of the stored procedure that you want to call.

Results

For example, suppose that you want to write one program, PROGY, that calls one of two versions of a stored procedure named PROCX. The load module for both stored procedures is named SUMMOD. Each version of SUMMOD is in a different load library. The stored procedures run in different WLM environments, and the startup JCL for each WLM environment includes a STEPLIB concatenation that specifies the correct load library for the stored procedure module.

First, define the two stored procedures in different schemas and different WLM environments:

```
CREATE PROCEDURE TEST.PROCX(IN V1 INTEGER, OUT V2 CHAR(9))
  LANGUAGE C
  EXTERNAL NAME SUMMOD
  WLM ENVIRONMENT TESTENV;
```

```
CREATE PROCEDURE PROD.PROCX(IN V1 INTEGER, OUT V2 CHAR(9))
  LANGUAGE C
  EXTERNAL NAME SUMMOD
  WLM ENVIRONMENT PRODENV;
```

When you write CALL statements for PROCX in program PROGY, use the unqualified form of the stored procedure name:

```
CALL PROCX(V1,V2);
```

Bind two plans for PROGY. In one BIND statement, specify PATH(TEST). In the other BIND statement, specify PATH(PROD).

To call TEST.PROCX, execute PROGY with the plan that you bound with PATH(TEST). To call PROD.PROCX, execute PROGY with the plan that you bound with PATH(PROD).

Invoking multiple instances of a stored procedure

Your application program can issue multiple CALL statements to the same local or remote stored procedure. Assume that your stored procedure returns result sets and the calling application leaves those result sets open before the next call to that same stored procedure. In that case, each CALL statement invokes a unique instance of the stored procedure.

About this task

When you invoke multiple instances of a stored procedure, each instance runs serially within the same Db2 thread and opens its own result sets. These multiple calls invoke multiple instances of any packages that are invoked while running the stored procedure. These instances are invoked at either the same or different level of nesting under one Db2 connection or thread.

For local stored procedures that issue remote SQL, instances of the applications are created at the remote server site. These instances are created regardless of whether result sets exist or are left open between calls.

If you call too many instances of a stored procedure or if you open too many cursors, Db2 storage shortages and EDM POOL FULL conditions might occur. If the stored procedure issues remote SQL statements to another Db2 server, these conditions can occur at both the Db2 client and at the Db2 server.

Procedure

To invoke multiple instances of a stored procedure:

1. To optimize storage usage and prevent storage shortages, ensure that you specify appropriate values for the following two subsystem parameters:

   MAX_ST_PROC
   Controls the maximum number of stored procedure instances that you can call within the same thread.

   MAX_NUM_CUR
   Controls the maximum number of cursors that can be opened by the same thread.

   When either of the values from these subsystem parameters is exceeded while an application is running, the CALL statement or the OPEN statement receives SQLCODE -904.
2. In your application, issue CALL statements to the stored procedure.
3. In the calling application for the stored procedure, close the result sets and issue frequent commits. Even read-only applications should perform these actions.

   Applications that fail to close result sets or issue an adequate number of commits might terminate abnormally with Db2 storage shortage and EDM POOL FULL conditions.

Related reference

MAX OPEN CURSORS field (MAX_NUM_CUR subsystem parameter) (Db2 Installation and Migration)
MAX STORED PROCS field (MAX_ST_PROC subsystem parameter) (Db2 Installation and Migration)
CALL (Db2 SQL)

Designating the active version of a native SQL procedure

When a native SQL procedure is called, Db2 uses the version that is designated as the active version.

About this task

When you create a native SQL procedure, that first version is by default the active version. If you create additional versions of a stored procedure, you can designate another version to be the active version.

Exception: If an existing active version is still being used by a process, the new active version is not used until the next call to that procedure.

Procedure

To designate the active version of a native SQL procedure, issue an ALTER PROCEDURE statement with the following items:

- The name of the native SQL procedure for which you want to change the active version.
- The ACTIVATE VERSION clause with the name of the version that you want to be active.

When the ALTER statement is committed, the new version of the procedure becomes the active version and is used by the next call for that procedure.

Example

The following ALTER PROCEDURE statement makes version V2 of the UPDATE_BALANCE procedure the active version.

```
ALTER PROCEDURE UPDATE_BALANCE
ACTIVATE VERSION V2;
```

Temporarily overriding the active version of a native SQL procedure

If you want a particular call to a native SQL procedure to use a version other than the active version, you can temporarily override the active version. Such an override might be helpful when you are testing a new version of a native SQL procedure.

About this task

Recommendation: If you want all calls to a native SQL procedure to use a particular version, do not temporarily override the active version in every call. Instead, make that version the active version. Otherwise, performance might be slower.

Procedure

To temporarily override the active version of a native SQL procedure, specify the following statements in your program:

1. The SET CURRENT ROUTINE VERSION statement with the name of the version of the procedure that you want to use. If the specified version does not exist, the active version is used.
2. The CALL statement with the name of the procedure.

Example

The following CALL statement invokes version V1 of the UPDATE_BALANCE procedure, regardless of what the current active version of that procedure is.

```
SET CURRENT ROUTINE VERSION = V1;
SET procname = 'UPDATE_BALANCE';
CALL :procname USING DESCRIPTOR :x;
```

Specifying the number of stored procedures that can run concurrently

Multiple stored procedures can run concurrently, each under its own z/OS task control block (TCB). The z/OS Workload Manager (WLM) manages how many concurrent stored procedures can run in an address space. The number of concurrent stored procedures in an address space cannot exceed the value of the NUMTCB field that was specified on the DSNTIPX installation panel, during Db2 installation.

Procedure

You can override that value in the following ways:

- Edit the JCL procedures that start stored procedures address spaces, and modify the value of the NUMTCB parameter.
- Specify the following parameter in the Start Parameters field of the Create An Application Environment panel when you set up a WLM application environment:

  ```
  NUMTCB=number-of-TCBs
  ```

  Special cases:

  – For REXX stored procedures, you must set the NUMTCB parameter to 1.

  – Stored procedures that invoke utilities can invoke only one utility at a time in a single address space. Consequently, the value of the NUMTCB parameter is forced to 1 for those procedures.

Related concepts

Installation step 19: Configure Db2 for running stored procedures and user-defined functions (Db2 Installation and Migration)

Migration step 22: Configure Db2 for running stored procedures and user-defined functions (optional) (Db2 Installation and Migration)

Related tasks

Maximizing the number of procedures or functions that run in an address space (Db2 Performance)

Retrieving the procedure status

When an SQL procedure returns control to the calling program, it also returns the procedure status. The status is an integer value that indicates the success of the procedure.

About this task

Db2 sets the status to 0 or -1 depending on the value of the SQLCODE. Alternatively, an SQL procedure can set the integer status value by using the RETURN statement. In this case, Db2 sets the SQLCODE in the SQLCA to 0.

Procedure

To retrieve the procedure status, perform one of the following actions in the calling program:

- Issue the GET DIAGNOSTICS statement with the DB2_RETURN_STATUS item. The specified host variable in the GET DIAGNOSTICS statement is set to one of the following values:

  0

  This value indicates that the procedure returned with an SQLCODE that is greater or equal to zero. You can access the value directly from the SQLCA by retrieving the value of SQLERRD(1). For C applications, retrieve SQLERRD[0].

  -1

  This value indicates that the procedure returned with an SQLCODE that is less than zero. In this case, the SQLERRD(1) value in the SQLCA is not set. Db2 returns -1 only.

  n

  Any value other than 0 or -1 is the return value that was explicitly set in the procedure with the RETURN statement.

  For example, the following SQL code creates an SQL procedure that is named TESTIT, which calls another SQL procedure that is named TRYIT. The TRYIT procedure returns a status value. The TESTIT procedure retrieves that value with the DB2_RETURN_STATUS item of the GET DIAGNOSTICS statement.

  ```
  CREATE PROCEDURE TESTIT ()
    LANGUAGE SQL
    A1:BEGIN
    DECLARE RETVAL INTEGER DEFAULT 0;
      ...
      CALL TRYIT;
      GET DIAGNOSTICS RETVAL = DB2_RETURN_STATUS;
      IF RETVAL <> 0 THEN
        ...
        LEAVE A1;
      ELSE
        ...
      END IF;
    END A1
  ```

- Retrieve the value of SQLERRD(1) in the SQLCA. For C applications, retrieve SQLERRD[0]. This field contains the integer value that was set by the RETURN statement in the SQL procedure. This method is not applicable if the status was set by Db2.

Related concepts

SQL communication area (SQLCA) (Db2 SQL)

Related reference

GET DIAGNOSTICS (Db2 SQL)

Writing a program to receive the result sets from a stored procedure

You can write a program to receive results set from a stored procedure for either a fixed number of result sets, for which you know the contents, or a variable number of result sets, for which you do not know the contents.

About this task

A program for a fixed number of result sets is simpler to write than a program for a variable number of result sets. However, if you write a program for a variable number of result sets, you do not need to make modifications to the program if the stored procedure changes.

If your program calls an SQL procedure that returns result sets, you must write the program for a fixed number of result sets.

In the following steps, you do not need to connect to the remote location when you execute these statements:

- DESCRIBE PROCEDURE
- ASSOCIATE LOCATORS
- ALLOCATE CURSOR
- DESCRIBE CURSOR
- FETCH
- CLOSE

Procedure

To write a program to receive the result sets from a stored procedure:

1. Declare a locator variable for each result set that is to be returned.

   If you do not know how many result sets are to be returned, declare enough result set locators for the maximum number of result sets that might be returned.
2. Call the stored procedure and check the SQL return code.

   If the SQLCODE from the CALL statement is +466, the stored procedure has returned result sets.
3. Determine how many result sets the stored procedure is returning.

   If you already know how many result sets the stored procedure returns, skip this step.

   Use the SQL statement DESCRIBE PROCEDURE to determine the number of result sets. DESCRIBE PROCEDURE places information about the result sets in an SQLDA. Make this SQLDA large enough to hold the maximum number of result sets that the stored procedure might return. When the DESCRIBE PROCEDURE statement completes, the fields in the SQLDA contain the following values:

   - SQLD contains the number of result sets that are returned by the stored procedure.
   - Each SQLVAR entry gives the following information about a result set:
     - The SQLNAME field contains the name of the SQL cursor that is used by the stored procedure to return the result set.
     - The SQLIND field contains the value -1, which indicates that no estimate of the number of rows in the result set is available.
     - The SQLDATA field contains the value of the result set locator, which is the address of the result set.
4. Link result set locators to result sets by performing one of the following actions:

   - Use the ASSOCIATE LOCATORS statement. You must embed this statement in an application or SQL procedure. The ASSOCIATE LOCATORS statement assigns values to the result set locator variables.

If you specify more locators than the number of result sets that are returned, Db2 ignores the extra locators.

- If you executed the DESCRIBE PROCEDURE statement previously, the result set locator values are in the SQLDATA fields of the SQLDA. You can copy the values from the SQLDATA fields to the result set locators manually, or you can execute the ASSOCIATE LOCATORS statement to do it for you.

The stored procedure name that you specify in an ASSOCIATE LOCATORS statement or DESCRIBE PROCEDURE statement must match the stored procedure name in the CALL statement as follows:

- If the name is unqualified in the CALL statement, do not qualify it.
- If the name is qualified with a schema name in the CALL statement, qualify it with the schema name.
- If the name is qualified with a location name and schema name in the CALL statement, qualify it with a location name and schema name.

5. Allocate cursors for fetching rows from the result sets.

   Use the SQL statement ALLOCATE CURSOR to link each result set with a cursor. Execute one ALLOCATE CURSOR statement for each result set. The cursor names can differ from the cursor names in the stored procedure.

   To use the ALLOCATE CURSOR statement, you must embed it in an application or SQL procedure.

6. Determine the contents of the result sets.

   If you already know the format of the result set, skip this step.

   Use the SQL statement DESCRIBE CURSOR to determine the format of a result set and put this information in an SQLDA. For each result set, you need an SQLDA that is big enough to hold descriptions of all columns in the result set.

   You can use DESCRIBE CURSOR for only those cursors for which you executed ALLOCATE CURSOR previously.

   After you execute DESCRIBE CURSOR, if the cursor for the result set is declared WITH HOLD, the high-order bit of byte 8 of field SQLDAID in the SQLDA is set to 1.

7. Fetch rows from the result sets into host variables by using the cursors that you allocated with the ALLOCATE CURSOR statements.

   Fetching rows from a result set is the same as fetching rows from a table.

   If you executed the DESCRIBE CURSOR statement, perform the following steps before you fetch the rows:

   a. Allocate storage for host variables and indicator variables. Use the contents of the SQLDA from the DESCRIBE CURSOR statement to determine how much storage you need for each host variable.
   b. Put the address of the storage for each host variable in the appropriate SQLDATA field of the SQLDA.
   c. Put the address of the storage for each indicator variable in the appropriate SQLIND field of the SQLDA.

Example

The following examples show C language code that accomplishes each of these steps. Coding for other languages is similar.

The following example demonstrates how to receive result sets when you know how many result sets are returned and what is in each result set.

```
/*************************************************************/
/* Declare result set locators.  For this example,           */
/* assume you know that two result sets will be returned.    */
/* Also, assume that you know the format of each result set. */
/*************************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
   static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2;
  EXEC SQL END DECLARE SECTION;
```

```
  ⋮
/*************************************************************/
/* Call stored procedure P1.                                 */
/* Check for SQLCODE +466, which indicates that result sets  */
/* were returned.                                            */
/*************************************************************/
  EXEC SQL CALL P1(:parm1, :parm2, ...);
  if(SQLCODE==+466)
  {
  /*************************************************************/
  /* Establish a link between each result set and its          */
  /* locator using the ASSOCIATE LOCATORS.                     */
  /*************************************************************/
    EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2) WITH PROCEDURE P1;
    ⋮
  /*************************************************************/
  /* Associate a cursor with each result set.                  */
  /*************************************************************/
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
    EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;
  /*************************************************************/
  /* Fetch the result set rows into host variables.            */
  /*************************************************************/
    while(SQLCODE==0)
    {
      EXEC SQL FETCH C1 INTO :order_no, :cust_no;
    ⋮
    }
    while(SQLCODE==0)
    {
      EXEC SQL FETCH C2 :order_no, :item_no, :quantity;
    ⋮
    }
  }
```

The following example demonstrates how to receive result sets when you do not know how many result sets are returned or what is in each result set.

```
/*************************************************************/
/* Declare result set locators.  For this example,           */
/* assume that no more than three result sets will be        */
/* returned, so declare three locators.  Also, assume        */
/* that you do not know the format of the result sets.       */
/*************************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
   static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;
  EXEC SQL END DECLARE SECTION;
  ⋮
```

```
/*************************************************************/
/* Call stored procedure P2.                                 */
/* Check for SQLCODE +466, which indicates that result sets  */
/* were returned.                                            */
/*************************************************************/
  EXEC SQL CALL P2(:parm1, :parm2, ...);
  if(SQLCODE==+466)
  {
  /*************************************************************/
  /* Determine how many result sets P2 returned, using the     */
  /* statement DESCRIBE PROCEDURE.  :proc_da is an SQLDA       */
  /* with enough storage to accommodate up to three SQLVAR     */
  /* entries.                                                  */
  /*************************************************************/
    EXEC SQL DESCRIBE PROCEDURE P2 INTO :proc_da;
    ⋮
  /*************************************************************/
  /* Now that you know how many result sets were returned,     */
  /* establish a link between each result set and its          */
  /* locator using the ASSOCIATE LOCATORS.  For this example,  */
  /* we assume that three result sets are returned.            */
  /*************************************************************/
    EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2, :loc3) WITH PROCEDURE P2;
    ⋮
  /*************************************************************/
  /* Associate a cursor with each result set.                  */
  /*************************************************************/
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
```

```
    EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;
    EXEC SQL ALLOCATE C3 CURSOR FOR RESULT SET :loc3;
```

```
  /*************************************************************/
  /* Use the statement DESCRIBE CURSOR to determine the        */
  /* format of each result set.                                */
  /*************************************************************/
    EXEC SQL DESCRIBE CURSOR C1 INTO :res_da1;
    EXEC SQL DESCRIBE CURSOR C2 INTO :res_da2;
    EXEC SQL DESCRIBE CURSOR C3 INTO :res_da3;
    ⋮
  /*************************************************************/
  /* Assign values to the SQLDATA and SQLIND fields of the     */
  /* SQLDAs that you used in the DESCRIBE CURSOR statements.   */
  /* These values are the addresses of the host variables and  */
  /* indicator variables into which DB2 will put result set    */
  /* rows.                                                     */
  /*************************************************************/
    ⋮
  /*************************************************************/
  /* Fetch the result set rows into the storage areas          */
  /* that the SQLDAs point to.                                 */
  /*************************************************************/
    while(SQLCODE==0)
    {
      EXEC SQL FETCH C1 USING :res_da1;
    ⋮
    }
    while(SQLCODE==0)
    {
      EXEC SQL FETCH C2 USING :res_da2;
    ⋮
    }
    while(SQLCODE==0)
    {
      EXEC SQL FETCH C3 USING :res_da3;
    ⋮
    }
  }
```

The following example demonstrates how you can use an SQL procedure to receive result sets. The logic assumes that no handler exists to intercept the +466 SQLCODE, such as `DECLARE CONTINUE HANDLER FOR SQLWARNING .....` Such a handler causes SQLCODE to be reset to zero. Then the test for IF SQLCODE = 466 is never true and the statements in the IF body are never executed.

```
DECLARE RESULT1 RESULT_SET_LOCATOR VARYING;
 DECLARE RESULT2 RESULT_SET_LOCATOR VARYING;
 DECLARE AT_END, VAR1, VAR2 INT DEFAULT 0;
 DECLARE SQLCODE INTEGER DEFAULT 0;
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET AT_END = 99;
 SET TOTAL1 = 0;
 SET TOTAL2 = 0;
 CALL TARGETPROCEDURE();
 IF SQLCODE = 466 THEN
   ASSOCIATE RESULT SET LOCATORS(RESULT1,RESULT2)
      WITH PROCEDURE SPDG3091;
   ALLOCATE RSCUR1 CURSOR FOR RESULT1;
   ALLOCATE RSCUR2 CURSOR FOR RESULT2;
   WHILE AT_END = 0 DO
     FETCH RSCUR1 INTO VAR1;
     SET TOTAL1 = TOTAL1 + VAR1;
     SET VAR1 = 0;   /* Reset so the last value fetched is not added after AT_END
*/
   END WHILE;
   SET AT_END = 0;  /* Reset for next loop */
   WHILE AT_END = 0 DO
     FETCH RSCUR2 INTO VAR2;
     SET TOTAL2 = TOTAL2 + VAR2;
     SET VAR2 = 0;   /* Reset so the last value fetched is not added after AT_END
*/
   END WHILE;
 END IF;
```

Related concepts

Example programs that call stored procedures

Examples can be used as models when you write applications that call stored procedures. In addition, *prefix*.SDSNSAMP contains sample jobs DSNTEJ6P and DSNTEJ6S and programs DSN8EP1 and DSN8EP2, which you can run.

Related reference

ALLOCATE CURSOR (Db2 SQL)
ASSOCIATE LOCATORS (Db2 SQL)
CALL (Db2 SQL)
DESCRIBE CURSOR (Db2 SQL)
DESCRIBE PROCEDURE (Db2 SQL)
SQL descriptor area (SQLDA) (Db2 SQL)

Chapter 6. Coding methods for distributed data

You can access distributed data by using three-part table names or explicit connect statements.

Introductory concepts

Distributed data (Introduction to Db2 for z/OS)
Effects of distributed data on programming (Introduction to Db2 for z/OS)
Distributed data access (Introduction to Db2 for z/OS)

Three-part table names are described in "Accessing distributed data by using three-part table names" on page 773. Explicit connect statements are described in "Accessing distributed data by using explicit CONNECT statements" on page 775.

These two methods of coding applications for distributed access are illustrated by the following example.

Spiffy Computer has a main project table that supplies information about all projects that are currently active throughout the company. Spiffy has several branches in various locations around the world, each a Db2 location that maintains a copy of the project table named DSN8B10.PROJ. The main branch location occasionally inserts data into all copies of the table. The application that makes the inserts uses a table of location names. For each row that is inserted, the application executes an INSERT statement in DSN8B10.PROJ for each location.

Copying a table from a remote location

To copy a table from one location to another, you can either write your own application program or use the Db2 DataPropagator product.

Related concepts

Monitoring Db2 in distributed environments (Db2 Performance)

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Accessing distributed data by using three-part table names

You can use three-part table names to access data at a remote location through DRDA access.

When you use three-part table names, you must create copies of the package that you used at the local site at all possible remote locations that could be accessed by the three-part table name references. You must also explicitly or generically specify remote packages in the PKLIST of the PLAN that is used by the application.

Recommendation: Always use an alias, which resolves to a three-part table name, rather than specifying a specific three-part table name in an SQL statement. Using an alias will permit you to physically move the location of the table as needed. By using an alias, you can drop and re-create the alias by specifying the tables's new remote location and then rebind the packages of the application.

In a three-part table name, the first part denotes the location. The local Db2 makes and breaks an implicit connection to a remote server as needed.

When a three-part name is parsed and forwarded to a remote location, any special register settings are automatically propagated to remote server. This allows the SQL statements to process the same way no matter at what site a statement is run.

Example

The following example assumes that all systems involved implement two-phase commit. This example suggests updating several systems in a loop and ending the unit of work by committing only when the loop is complete. Updates are coordinated across the entire set of systems.

Spiffy's application uses a location name to construct a three-part table name in an INSERT statement. It then prepares the statement and executes it dynamically. The values to be inserted are transmitted to the remote location and substituted for the parameter markers in the INSERT statement.

The following overview shows how the application uses aliases for three-part names:

```
Read in the alias values
Do for all locations
     Read location name
     Set up statement to prepare
     Prepare statement
a     Execute statement
End loop
Commit
```

After the application obtains the next alias of a remote table to be inserted, For example, REGION1PROJ (which is the DSN8B10.PROJ table at location SAN_JOSE), it creates the following character string:

```
INSERT INTO REGION1PROJ VALUES (?,?,?,?,?,?,?,?)
```

The alias is created as follows:

```
CREATE ALIAS REGION1PROJ FOR SAN_JOSE.DSN8B10.PROJ
```

The application assigns the character string to the variable INSERTX and then executes these statements:

```
EXEC SQL
   PREPARE STMT1 FROM :INSERTX;
EXEC SQL
   EXECUTE STMT1 USING :PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,
                       :PRSTAFF, :PRSTDATE, :PRENDATE, :MAJPROJ;
```

The host variables for Spiffy's project table match the declaration for the sample project table.

To keep the data consistent at all locations, the application commits the work only when the loop has executed for all locations. Either every location has committed the INSERT or, if a failure has prevented any location from inserting, all other locations have rolled back the INSERT. (If a failure occurs during the commit process, the entire unit of work can be indoubt.)

Three-part names and multiple servers

Recommendation: Always use an asterisk (*) for the location name in a pklist. Never use the explicit location name unless you are sure that no other location could ever be accessed.

The following steps are recommended:

1. Bind the DBRM into a package at the local Db2.
2. Bind package copy at the first target site of the alias.
3. Bind package copy at the target site.

Related concepts

Considerations for binding packages at a remote location
When you bind packages at a remote location, you need to understand how the behavior of the remote packages differs from the behavior of local packages.

Aliases (Db2 SQL)

Synonyms (deprecated) (Db2 SQL)

Related tasks

Including dynamic SQL in your program
Dynamic SQL is prepared and executed while the program is running.

Related reference

Project table (DSN8B10.PROJ) (Introduction to Db2 for z/OS)

Accessing remote declared temporary tables by using three-part table names

You can access a remote declared temporary table by using a three-part name. However, if you combine explicit CONNECT statements and three-part names in your application, a reference to a remote declared temporary table must be a forward reference.

In a CREATE GLOBAL TEMPORARY TABLE or DECLARE GLOBAL TEMPORARY TABLE statement, you cannot specify an alias that resolves to a three-part name object at a remote location. You also cannot specify a three-part name object even if the location of the three-part name refers to the location where the object is being created or declared.

Example

You can perform the following series of actions, which includes a forward reference to a declared temporary table:

```
EXEC SQL CONNECT TO CHICAGO;              /* Connect to the remote site */
EXEC SQL
  DECLARE GLOBAL TEMPORARY TABLE T1 /* Define the temporary table */
  (CHARCOL CHAR(6) NOT NULL)             /* at the remote site         */
   ON COMMIT DROP TABLE;
EXEC SQL CONNECT RESET;                   /* Connect back to local site */
EXEC SQL INSERT INTO CHICAGO.SESSION.T1
  (VALUES 'ABCDEF');                      /* Access the temporary table*/
                              /* at the remote site (forward reference) */
```

However, you cannot perform the following series of actions, which includes a backward reference to the declared temporary table:

```
EXEC SQL
  DECLARE GLOBAL TEMPORARY TABLE T1 /* Define the temporary table */
  (CHARCOL CHAR(6) NOT NULL)             /* at the local site (ATLANTA)*/
   ON COMMIT DROP TABLE;
EXEC SQL CONNECT TO CHICAGO;              /* Connect to the remote site */
EXEC SQL INSERT INTO ATLANTA.SESSION.T1
  (VALUES 'ABCDEF');                      /* Cannot access temp table  */
                            /* from the remote site (backward reference)*/
```

Example using an alias

You can perform the following series of actions, which includes a forward reference to a declared temporary table using an alias. First you need to declare the alias at the requester. The name you give the alias must resolve to match the real name.

```
CREATE APPLT1 FOR CHICAGO.SESSION.T1
```

The CONNECT and DECLARE statements refer to the real declared temp table.

```
EXEC SQL CONNECT TO CHICAGO;
EXEC SQL DECLARE GLOBAL TEMPORARY TABLE T1
 (CHARCOL CHAR(6) NOT NULL)
  ON COMMIT DROP TABLE;
EXEC SQL CONNECT RESET;
EXEC SQL INSERT INTO APPLT1 VALUES ('ABCDEF');
```

Accessing distributed data by using explicit CONNECT statements

When you use explicit CONNECT statements to access distributed data, the application program explicitly connects to each new server.

About this task

You must bind the DBRMs for the SQL statements to be executed at the server to packages that reside at that server.

The following example assumes that all systems involved implement two-phase commit. This example suggests updating several systems in a loop and ending the unit of work by committing only when the loop is complete. Updates are coordinated across the entire set of systems.

In this example, Spiffy's application executes CONNECT for each server in turn, and the server executes INSERT. In this case, the tables to be updated each have the same name, although each table is defined at a different server. The application executes the statements in a loop, with one iteration for each server.

The application connects to each new server by means of a host variable in the CONNECT statement. CONNECT changes the special register CURRENT SERVER to show the location of the new server. The values to insert in the table are transmitted to a location as input host variables.

The following overview shows how the application uses explicit CONNECTs:

```
Read input values
Do for all locations
     Read location name
     Connect to location
     Execute insert statement
End loop
Commit
Release all
```

For example, the application inserts a new location name into the variable LOCATION_NAME and executes the following statements:

```
EXEC SQL
   CONNECT TO :LOCATION_NAME;
EXEC SQL
   INSERT INTO DSN8B10.PROJ VALUES (:PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,
                                    :PRSTAFF, :PRSTDATE, :PRENDATE, :MAJPROJ);
```

To keep the data consistent at all locations, the application commits the work only when the loop has executed for all locations. Either every location has committed the INSERT or, if a failure has prevented any location from inserting, all other locations have rolled back the INSERT. (If a failure occurs during the commit process, the entire unit of work can be indoubt.)

The host variables for Spiffy's project table match the declaration for the sample project table. LOCATION_NAME is a character-string variable of length 16.

Related reference

Project table (DSN8B10.PROJ) (Introduction to Db2 for z/OS)

Specifying a location alias name for multiple sites

You can override the location name that an application uses to access a server.

About this task

Db2 uses the DBALIAS value in the SYSIBM.LOCATIONS table to override the location name that an application uses to access a server.

For example, suppose that an employee database is deployed across two sites and that both sites make themselves known as location name EMPLOYEE. To access each site, insert a row for each site into SYSIBM.LOCATIONS with the location names SVL_EMPLOYEE and SJ_EMPLOYEE. Both rows contain EMPLOYEE as the DBALIAS value. When an application issues a CONNECT TO SVL_EMPLOYEE statement, Db2 searches the SYSIBM.LOCATIONS table to retrieve the location and network attributes of the database server. Because the DBALIAS value is not blank, Db2 uses the alias EMPLOYEE, and not the location name, to access the database.

If the application uses fully qualified object names in its SQL statements, Db2 sends the statements to the remote server without modification. For example, suppose that the application issues the statement SELECT * FROM SVL_EMPLOYEE.*authid.table* with the fully-qualified object name. However, Db2 accesses the remote server by using the EMPLOYEE alias. The remote server must identify itself as both SVL_EMPLOYEE and EMPLOYEE; otherwise, it rejects the SQL statement with a message indicating that the database is not found. If the remote server is Db2, the location SVL_EMPLOYEE might be defined as a location alias for EMPLOYEE. Db2 z/OS servers are defined with this alias by using the DDF ALIAS statement of the DSNJU003 change log inventory utility. Db2 locally executes any SQL statements that contain fully qualified object names if the high-level qualifier is the location name or any of its alias names.

Related reference

LOCATIONS catalog table (Db2 SQL)

DSNJU003 (change log inventory) (Db2 Utilities)

Releasing connections

When you connect to remote locations explicitly, you must also terminate those connections explicitly.

About this task

To break the connections, you can use the RELEASE statement. The RELEASE statement differs from the CONNECT statement in the following ways:

- While the CONNECT statement makes an immediate connection, the RELEASE statement **does not** immediately break a connection. The RELEASE statement labels connections for release at the next commit point. A connection that has been labeled for release is in the *release-pending state* and can still be used before the next commit point.
- While the CONNECT statement connects to exactly one remote system, you can use the RELEASE statement to specify a single connection or a set of connections for release at the next commit point.

Example

By using the RELEASE statement, you can place any of the following connections in the release-pending state:

- A specific connection that the next unit of work does not use:

```
EXEC SQL RELEASE SPIFFY1;
```

- The current SQL connection, whatever its location name:

```
EXEC SQL RELEASE CURRENT;
```

- All connections except the local connection:

```
EXEC SQL RELEASE ALL;
```

Transmitting mixed data

Mixed data is data that contains both character and graphic data.

About this task

If you transmit mixed data between your local system and a remote system, put the data in varying-length character strings instead of fixed-length character strings.

Converting mixed data: When ASCII MIXED data or Unicode MIXED data is converted to EBCDIC MIXED, the converted string is longer than the source string. An error occurs if that conversion is performed on a fixed-length input host variable. The remedy is to use a varying-length string variable with a maximum length that is sufficient to contain the expansion.

Identifying the server at run time

You can request the location name of the system to which you are connected.

About this task

The special register CURRENT SERVER contains the location name of the system you are connected to. You can assign that name to a host variable with a statement like this:

```
EXEC SQL SET :CS = CURRENT SERVER;
```

SQL limitations at dissimilar servers

When you execute SQL statements on a remote server that is running another Db2 family product, certain limitations exist. Generally, a program that uses DRDA access can use SQL statements and clauses that are supported by a remote server, even if they are not supported by the local server.

The following examples suggest what to expect from dissimilar servers:

- They support SELECT, INSERT, UPDATE, DELETE, DECLARE CURSOR, and FETCH, but details vary.

  Example: Db2 for Linux, UNIX, and Windows and DB2 for i support a form of INSERT that allows for multiple rows of input data. In this case, the VALUES clause is followed by multiple lists in parentheses. Each list represents the values to be inserted for a row of data. Db2 for z/OS does not support this form of INSERT.
- Data definition statements vary more widely.

  Example: Db2 for z/OS supports ROWID columns; Db2 for Linux, UNIX, and Windows does not support ROWID columns. Any data definition statements that use ROWID columns cannot run across all platforms.
- Statements can have different limits.

  Example: A query in Db2 for z/OS can have 750 columns; for other systems, the maximum is higher. But a query using 750 or fewer columns could execute in all systems.
- Some statements are not sent to the server but are processed completely by the requester. You cannot use those statements in a remote package even though the server supports them.
- In general, if a statement to be executed at a remote server contains host variables, a Db2 requester assumes them to be input host variables unless it supports the syntax of the statement and can determine otherwise. If the assumption is not valid, the server rejects the statement.

Related reference

Characteristics of SQL statements in Db2 for z/OS (Db2 SQL)

Support for executing long SQL statements in a distributed environment

A distributed application can send prepared SQL statements exceed 32 KB in size. If the statements exceed 32 KB in size, the server must support these long statements.

If a distributed application assigns an SQL statement to a DBCLOB (UTF-16) variable and sends the prepared statement to a remote server, the remote Db2 server converts it to UTF-8. If the remote server does not support UTF-8, the requester converts the statement to the system EBCDIC CCSID before sending it to the remote server.

Distributed queries against ASCII or Unicode tables

When you perform a distributed query, the server determines the encoding scheme of the result table.

When a distributed query against an ASCII or Unicode table arrives at the Db2 for z/OS server, the server indicates in the reply message that the columns of the result table contain ASCII or Unicode data, rather than EBCDIC data. The reply message also includes the CCSIDs of the data to be returned. The CCSID of data from a column is the CCSID that was in effect when the column was defined.

The encoding scheme in which Db2 returns data depends on two factors:

- The encoding scheme of the requesting system.

  If the requester is ASCII or Unicode, the returned data is ASCII or Unicode. If the requester is EBCDIC, the returned data is EBCDIC, even though it is stored at the server as ASCII or Unicode. However, if the SELECT statement that is used to retrieve the data contains an ORDER BY clause, the data displays in ASCII or Unicode order.
- Whether the application program overrides the CCSID for the returned data. The ways to do this are as follows:
  - For static SQL

    You can bind a plan or package with the ENCODING bind option to control the CCSIDs for all static data in that plan or package. For example, if you specify ENCODING(UNICODE) when you bind a package at a remote Db2 for z/OS system, the data that is returned in host variables from the remote system is encoded in the default Unicode CCSID for that system.
  - For static or dynamic SQL

    An application program can specify overriding CCSIDs for individual host variables in DECLARE VARIABLE statements.

    An application program that uses an SQLDA can specify an overriding CCSID for the returned data in the SQLDA. When the application program executes a FETCH statement, you receive the data in the CCSID that is specified in the SQLDA.

Related tasks

Setting the CCSID for host variables
All Db2 string data, other than binary data, has an encoding scheme and a coded character set ID (CCSID) associated with it. You can associate an encoding scheme and a CCSID with individual host variables. Any data in those host variable is then associated with that encoding scheme and CCSID.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Restrictions when using scrollable cursors to access distributed data

The restrictions that exist for scrollable cursors depend on what the requester and the server support.

If a Db2 for z/OS server processes an OPEN cursor statement for a scrollable cursor, and the OPEN cursor statement comes from a requester that does not support scrollable cursors, the Db2 for z/OS server returns an SQL error. However, if a stored procedure at the server uses a scrollable cursor to return a result set, the down-level requester can access data through that cursor. The Db2 for z/OS server converts the scrollable result set cursor to a non-scrollable cursor. The requester can retrieve the data using sequential FETCH statements.

Restrictions when using rowset-positioned cursors to access distributed data

The restrictions that exist for row-positioned cursors depend on what the requester and the server support.

If a Db2 for z/OS server processes an OPEN cursor statement for a rowset-positioned cursor, and the OPEN cursor statement comes from a requester that does not support rowset-positioned cursors, the Db2 for z/OS server returns an SQL error. However, if a stored procedure at the server uses a rowset-positioned cursor to return a result set, the down-level requester can access data through that cursor by using row-positioned FETCH statements.

IBM MQ with Db2

IBM MQ is a message handling system that enables applications to communicate in a distributed environment across different operating systems and networks.

IBM MQ handles the communication from one program to another by using application programming interfaces (APIs). You can use any of the following APIs to interact with the IBM MQ message handling system:

- Message Queue Interface (MQI)
- IBM MQ classes for Java
- IBM MQ classes for Java Message Service (JMS)

Db2 provides its own application programming interface to the WebSphere® MQ message handling system through a set of external user-defined functions, which are called Db2 MQ functions. You can use these functions in SQL statements to combine Db2 database access with IBM MQ message handling. The Db2 MQ functions use the MQI.

Related reference

The Message Queue Interface overview

IBM MQ messages

IBM MQ uses messages to pass information between applications.

Messages consist of the following parts:

- The message attributes, which identify the message and its properties.
- The message data, which is the application data that is carried in the message.

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures
You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

IBM MQ message handling

Conceptually, the IBM MQ message handling system takes a piece of information (the message) and sends it to its destination. MQ guarantees delivery, despite any network disruptions that might occur.

In IBM MQ, a destination is called a message queue, and a queue resides in a queue manager. Applications can put messages on queues or get messages from them.

Db2 communicates with the WebSphere message handling system through a set of external user-defined functions, which are called Db2 MQ functions. These functions use the MQI.

When you send a message, you must specify the following three components:

message data
: Defines what is sent from one program to another.

service
Defines where the message is going to or coming from. The parameters for managing a queue are defined in the service, which is typically defined by a system administrator. The complexity of the parameters in the service is hidden from the application program.

policy
Defines how the message is handled. Policies control such items as:

- The attributes of the message, for example, the priority.
- Options for send and receive operations, for example, whether an operation is part of a unit of work.

The default service and policy are set as part of defining the WebSphere MQ configuration for a particular installation of Db2. (This action is typically performed by a system administrator.) Db2 provides the default service Db2.DEFAULT.SERVICE and the default policy Db2.DEFAULT.POLICY.

Related tasks

Additional steps for enabling IBM MQ user-defined functions (Db2 Installation and Migration)

Related reference

IBM MQ home

IBM MQ message handling with the MQI

One way to send and receive WebSphere MQ messages from Db2 applications is to use the Db2 MQ functions that use MQI.

These MQI-based functions use the services and policies that are defined in two Db2 tables, SYSIBM.MQSERVICE_TABLE and SYSIBM.MQPOLICY_TABLE. These tables are user-managed and are typically created and maintained by a system administrator. Each table contains a row for the default service and policy that are provided by Db2.

The application program does not need know the details of the services and policies that are defined in these tables. The application need only specify which service and policy to use for each message that it sends and receives. The application specifies this information when it calls a Db2 MQ function.

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures
You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

Related reference

Db2 MQ tables
The Db2 MQ tables contain service and policy definitions that are used by the Message Queue Interface (MQI) based Db2 MQ functions. You must populate the Db2 MQ tables before you can use these MQI-based functions.

Db2 MQI services

A service describes a destination to which an application sends messages or from which an application receives messages. Db2 Message Queue Interface (MQI) services are defined in the Db2 table SYSIBM.MQSERVICE_TABLE.

The MQI-based Db2 MQ functions use the services that are defined in the Db2 table SYSIBM.MQSERVICE_TABLE. This table is user-managed and is typically created and maintained by a system administrator. This table contains a row for each defined service, including your customized services and the default service that is provided by Db2.

The application program does not need know the details of the defined services. When an application program calls an MQI-based Db2 MQ function, the program selects a service from SYSIBM.MQSERVICE_TABLE by specifying it as a parameter.

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures

You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

IBM MQ message handling
Conceptually, the IBM MQ message handling system takes a piece of information (the message) and sends it to its destination. MQ guarantees delivery, despite any network disruptions that might occur.

Related reference

Db2 MQ tables
The Db2 MQ tables contain service and policy definitions that are used by the Message Queue Interface (MQI) based Db2 MQ functions. You must populate the Db2 MQ tables before you can use these MQI-based functions.

Db2 MQI policies

A policy controls how the MQ messages are handled. Db2 Message Queue Interface (MQI) policies are defined in the Db2 table SYSIBM.MQPOLICY_TABLE.

The MQI-based Db2 MQ functions use the policies that are defined in the Db2 table SYSIBM.MQPOLICY_TABLE. This table is user-managed and is typically created and maintained by a system administrator. This table contains a row for each defined policy, including your customized policies and the default policy that is provided by Db2.

The application program does not need know the details of the defined policies. When an application program calls an MQI-based Db2 MQ function, the program selects a policy from SYSIBM.MQPOLICY_TABLE by specifying it as a parameter.

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures
You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

IBM MQ message handling
Conceptually, the IBM MQ message handling system takes a piece of information (the message) and sends it to its destination. MQ guarantees delivery, despite any network disruptions that might occur.

Related reference

Db2 MQ tables
The Db2 MQ tables contain service and policy definitions that are used by the Message Queue Interface (MQI) based Db2 MQ functions. You must populate the Db2 MQ tables before you can use these MQI-based functions.

Db2 MQ functions and Db2 MQ XML stored procedures

You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

The Db2 MQ functions support the following types of operations:

- Send and forget, where no reply is needed.
- Read or receive, where one or all messages are either read without removing them from the queue, or received and removed from the queue.
- Request and response, where a sending application needs a response to a request.
- Publish and subscribe, where messages are assigned to specific publisher services and are sent to queues. Applications that subscribe to the corresponding subscriber service can monitor specific messages.

You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue. You can send a request to a message queue and receive a response, and you can also publish messages to the IBM MQ publisher and subscribe to messages that have been published with specific topics. The Db2 MQ XML functions and stored procedures enable you to query XML documents and then publish the results to a message queue.

The Db2 MQ functions include scalar functions, table functions, and XML-specific functions. For each of these functions, you can call a version that uses the MQI. The function signatures are the same. However, the qualifying schema names are different. To call an MQI-based function, specify the schema name DB2MQ.

Requirement: Before you can call the version of these functions that uses MQI , you need to populate the Db2 MQ tables.

The following table describes the Db2 MQ scalar functions.

Table 126. Db2 MQ scalar functions

Scalar function	Description
MQREAD (*receive-service, service-policy*)	MQREAD returns a message in a VARCHAR variable from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the message from the head of the queue but instead returns it. If no messages are available to be returned, a null value is returned.
MQREADCLOB (*receive-service, service-policy*)	MQREADCLOB returns a message in a CLOB variable from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the message from the head of the queue but instead returns it. If no messages are available to be returned, a null value is returned.
MQRECEIVE (*receive-service, service-policy, correlation-id*)	MQRECEIVE returns a message in a VARCHAR variable from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the message from the queue. If *correlation-id* is specified, the first message with a matching correlation identifier is returned; if *correlation-id* is not specified, the message at the beginning of queue is returned. If no messages are available to be returned, a null value is returned.
MQRECEIVECLOB (*receive-service, service-policy, correlation-id*)	MQRECEIVECLOB returns a message in a CLOB variable from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the message from the queue. If *correlation-id* is specified, the first message with a matching correlation identifier is returned; if *correlation-id* is not specified, the message at the head of queue is returned. If no messages are available to be returned, a null value is returned.
MQSEND (*send-service, service-policy, msg-data, correlation-id*)	MQSEND sends the data in a VARCHAR or CLOB variable *msg-data* to the MQ location specified by *send-service*, using the policy defined in *service-policy*. An optional user-defined message correlation identifier can be specified by *correlation-id*. The return value is 1 if successful or 0 if not successful.

Notes:

1. You can send or receive messages in VARCHAR variables or CLOB variables. The maximum length for a message in a VARCHAR variable is 32 KB. The maximum length for a message in a CLOB variable is 2 MB.

The following table describes the MQ table functions that Db2 can use.

Table 127. Db2 MQ table functions

Table function	Description
MQREADALL (*receive-service, service-policy, num-rows*)	MQREADALL returns a table that contains the messages and message metadata in VARCHAR variables from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the messages from the queue. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.
MQREADALLCLOB (*receive-service, service-policy, num-rows*)	MQREADALLCLOB returns a table that contains the messages and message metadata in CLOB variables from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the messages from the queue. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.
MQRECEIVEALL (*receive-service, service-policy, correlation-id, num-rows*)	MQRECEIVEALL returns a table that contains the messages and message metadata in VARCHAR variables from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the messages from the queue. If *correlation-id* is specified, only those messages with a matching correlation identifier are returned; if *correlation-id* is not specified, all available messages are returned. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.
MQRECEIVEALLCLOB (*receive-service, service-policy, correlation-id, num-rows*)	MQRECEIVEALLCLOB returns a table that contains the messages and message metadata in CLOB variables from the MQ location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the messages from the queue. If *correlation-id* is specified, only those messages with a matching correlation identifier are returned; if *correlation-id* is not specified, all available messages are returned. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.

Notes:

1. You can send or receive messages in VARCHAR variables or CLOB variables. The maximum length for a message in a VARCHAR variable is 32 KB. The maximum length for a message in a CLOB variable is 2 MB.
2. The first column of the result table of a Db2 MQ table function contains the message.

Related tasks

Additional steps for enabling IBM MQ user-defined functions (Db2 Installation and Migration)

Related reference

Procedures that are supplied with Db2 (Db2 SQL)
MQREADALL (Db2 SQL)
MQREADALLCLOB (Db2 SQL)
MQRECEIVEALL (Db2 SQL)
MQRECEIVEALLCLOB (Db2 SQL)
The Message Queue Interface overview

Generating XML documents from existing tables and sending them to an MQ message queue

You can send data from a Db2 table to the MQ message queue. First put the data in an XML document and then send that document to the message queue.

Procedure

To generate XML documents from existing tables and send them to an MQ message queue:

1. Compose an XML document by using the Db2 XML publishing functions.
2. Cast the XML document to type VARCHAR or CLOB.
3. Send the document to an MQ message queue by using the appropriate Db2 MQ function.

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures
You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

Functions for constructing XML values (Db2 Programming for XML)

Shredding XML documents from an MQ message queue

When you retrieve XML data from an MQ message queue, you can shred that data into Db2 tables for easy retrievability.

About this task

Procedure

To shred XML documents from an MQ message queue:

1. Retrieve the XML document from an MQ message queue by using the appropriate MQ function.
2. Shred the retrieved message to Db2 tables by using the XML decomposition stored procedure (XDBDECOMPXML).

Related concepts

Db2 MQ functions and Db2 MQ XML stored procedures
You can use the Db2 MQ functions and stored procedures to send messages to a message queue or to receive messages from the message queue.

Db2 MQ tables

The Db2 MQ tables contain service and policy definitions that are used by the Message Queue Interface (MQI) based Db2 MQ functions. You must populate the Db2 MQ tables before you can use these MQI-based functions.

The Db2 MQ tables are SYSIBM.MQSERVICE_TABLE and SYSIBM.MQPOLICY_TABLE. These tables are user-managed. You need to create them during the installation or migration process. Installation job DSNTIJRT creates these tables with one default row in each table.

If you previously used the AMI-based Db2 MQ functions, you used AMI configuration files instead of these tables. To use the MQI-based Db2 MQ functions, you need to move the data from those configuration files to the Db2 tables SYSIBM.MQSERVICE_TABLE and SYSIBM.MQPOLICY_TABLE .

The following table describes the columns for SYSIBM.MQSERVICE_TABLE.

Column name	Description
SERVICENAME	This column contains the service name, which is an optional input parameter of the MQ functions. This column is the primary key for the SYSIBM.MQSERVICE_TABLE table.
QUEUEMANAGER	This column contains the name of the queue manager where the MQ functions are to establish a connection.
INPUTQUEUE	This column contains the name of the queue from which the MQ functions are to send and retrieve messages.
CODEDCHARSETID	This column contains the character set identifier for character data in the messages that are sent and received by the MQ functions. This column corresponds to the CodedCharSetId field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the CodedCharSetId field. The default value for this column is 0, which sets the CodedCharSetId field of the MQMD to the value MQCCSI_Q_MGR.
ENCODING	This column contains the encoding value for the numeric data in the messages that are sent and received by the MQ functions. This column corresponds to the Encoding field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Encoding field. The default value for this column is 0, which sets the Encoding field in the MQMD to the value MQENC_NATIVE.
DESCRIPTION	This column contains the description of the service.

Table 128. SYSIBM.MQSERVICE_TABLE column descriptions

The following table describes the columns for SYSIBM.MQPOLICY_TABLE.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions

Column name	Description
POLICYNAME	This column contains the policy name, which is an optional input parameter of the MQ functions. This column is the primary key for the SYSIBM.MQPOLICY_TABLE table.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)	
Column name	**Description**
SEND_PRIORITY	This column contains the priority of the message. This column corresponds to the Priority field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Priority field. The default value for this column is -1, which sets the Priority field in the MQMD to the value MQQPRI_PRIORITY_AS_Q_DEF.
SEND_PERSISTENCE	This column indicates whether the message persists despite any system failures or instances of restarting the queue manager. This column corresponds to the Persistence field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Persistence field. This column can have the following values: **Q** Sets the Persistence field in the MQMD to the value MQPER_PERSISTENCE_AS_Q_DEF. This value is the default. **Y** Sets the Persistence field in the MQMD to the value MQPER_PERSISTENT. **N** Sets the Persistence field in the MQMD to the value MQPER_NOT_ PERSISTENT.
SEND_EXPIRY	This column contains the message expiration time, in tenths of a second. This column corresponds to the Expiry field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Expiry field. The default value is -1, which sets the Expiry field to the value MQEI_UNLIMITED.
SEND_RETRY_COUNT	This column contains the number of times that the MQ function is to try to send a message if the procedure fails. The default value is 5.
SEND_RETRY_INTERVAL	This column contains the interval, in milliseconds, between each attempt to send a message. The default value is 1000.

Column name	Description
SEND_NEW_CORRELID	This column specifies how the correlation identifier is to be set if a correlation identifier is not passed as an input parameter in the MQ function. The correlation identifier is set in the CorrelId field in the message descriptor structure (MQMD). This column can have one of the following values: **N** Sets the CorrelId field in the MQMD to binary zeros. This value is the default. **Y** Specifies that the queue manager is to generate a new correlation identifier and set the CorrelId field in the MQMD to that value. This 'Y' value is equivalent to setting the MQPMO_NEW_CORREL_ID option in the Options field in the put message options structure (MQPMO).
SEND_RESPONSE_MSGID	This column specifies how the MsgId field in the message descriptor structure (MQMD) is to be set for report and reply messages. This column corresponds to the Report field in the MQMD. MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Sets the MQRO_NEW_MSG_ID option in the Report field in the MQMD. This value is the default. **P** Sets the MQRO_PASS_MSG_ID option in the Report field in the MQMD.
SEND_RESPONSE_CORRELID	This column specifies how the CorrelID field in the message descriptor structure (MQMD) is to be set for report and reply messages. This column corresponds to the Report field in the MQMD. MQ functions use the value in this column to set the Report field. This column can have one of the following values: **C** Sets the MQRO_COPY_MSG_ID_TO_CORREL_ID option in the Report field in the MQMD. This value is the default. **P** Sets the MQRO_PASS_CORREL_ID option in the Report field in the MQMD.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Column name	Description
SEND_EXCEPTION_ACTION	This column specifies what to do with the original message when it cannot be delivered to the destination queue. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **Q** Sets the MQRO_DEAD_LETTER_Q option in the Report field in the MQMD. This value is the default. **D** Sets the MQRO_DISCARD_MSG option in the Report field in the MQMD. **P** Sets the MQRO_PASS_DISCARD_AND_EXPIRY option in the Report field in the MQMD.
SEND_REPORT_EXCEPTION	This column specifies whether an exception report message is to be generated when a message cannot be delivered to the specified destination queue and if so, what that report message should contain. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Specifies that an exception report message is not to be generated. No options in the Report field are set. This value is the default. **E** Sets the MQRO_EXCEPTION option in the Report field in the MQMD. **D** Sets the MQRO_EXCEPTION_WITH_DATA option in the Report field in the MQMD. **F** Sets the MQRO_EXCEPTION_WITH_FULL_DATA option in the Report field in the MQMD.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Column name	Description
SEND_REPORT_COA	This column specifies whether the queue manager is to send a confirm-on-arrival (COA) report message when the message is placed in the destination queue, and if so, what that COA message is to contain. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Specifies that a COA message is not to be sent. No options in the Report field are set. This value is the default **C** Sets the MQRO_COA option in the Report field in the MQMD **D** Sets the MQRO_COA_WITH_DATA option in the Report field in the MQMD. **F** Sets the MQRO_COA_WITH_FULL_DATA option in the Report field in the MQMD.
SEND_REPORT_COD	This column specifies whether the queue manager is to send a confirm-on-delivery (COD) report message when an application retrieves and deletes a message from the destination queue, and if so, what that COD message is to contain. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Specifies that a COD message is not to be sent. No options in the Report field are set. This value is the default. **C** Sets the MQRO_COD option in the Report field in the MQMD. **D** Sets the MQRO_COD_WITH_DATA option in the Report field in the MQMD. **F** Sets the MQRO_COD_WITH_FULL_DATA option in the Report field in the MQMD.

Column name	Description
SEND_REPORT_EXPIRY	This column specifies whether the queue manager is to send an expiration report message if a message is discarded before it is delivered to an application, and if so, what that message is to contain. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Specifies that an expiration report message is not to be sent. No options in the Report field are set.This value is the default. **C** Sets the MQRO_EXPIRATION option in the Report field in the MQMD. **D** Sets the MQRO_EXPIRATION_WITH_DATA option in the Report field in the MQMD. **F** Sets the MQRO_EXPIRATION_WITH_FULL_DATA option in the Report field in the MQMD.
SEND_REPORT_ACTION	This column specifies whether the receiving application sends a positive action notification (PAN), a negative action notification (NAN), or both. This column corresponds to the Report field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the Report field. This column can have one of the following values: **N** Specifies that neither notification is to be sent. No options in the Report field are set. This value is the default. **P** Sets the MQRO_PAN option in the Report field in the MQMD. **T** Sets the MQRO_NAN option in the Report field in the MQMD. **B** Sets both the MQRO_PAN and MQRO_NAN options in the Report field in the MQMD.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Column name	Description
SEND_MSG_TYPE	This column contains the type of message. This column corresponds to the MsqType field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the MsqType field. This column can have one of the following values: **DTG** Sets the MsgType field in the MQMD to MQMT_DATAGRAM. This value is the default. **REQ** Sets the MsgType field in the MQMD to MQMT_REQUEST. **RLY** Sets the MsgType field in the MQMD to MQMT_REPLY. **RPT** Sets the MsgType field in the MQMD to MQMT_REPORT.
REPLY_TO_Q	This column contains the name of the message queue to which the application that issued the MQGET call is to send reply and report messages. This column corresponds to the ReplyToQ field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the ReplyToQ field. The default value for this column is SAME AS INPUT_Q, which sets the name to the queue name that is defined in the service that was used for sending the message. If no service was specified, the name is set to DB2MQ_DEFAULT_Q, which is the name of the input queue for the default service.
REPLY_TO_QMGR	This column contains the name of the queue manager to which the reply and report messages are to be sent. This column corresponds to the ReplyToQMgr field in the message descriptor structure (MQMD). MQ functions use the value in this column to set the ReplyToQMgr field. The default value for this column is SAME AS INPUT_QMGR, which sets the name to the queue manager name that is defined in the service that was used for sending the message. If no service was specified, the name is set to the name of the queue manager for the default service.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)

Column name	Description
RCV_WAIT_INTERVAL	This column contains the time, in milliseconds, that Db2 is to wait for messages to arrive in the queue. This column corresponds to the WaitInterval field in the get message options structure (MQGMO). MQ functions use the value in this column to set the WaitInterval field. The default is 10.
RCV_CONVERT	This column indicates whether to convert the application data in the message to conform to the CodedCharSetId and Encoding values of the specified MQ service. This column corresponds to the Options field in the get message options structure (MQGMO). MQ functions use the value in this column to set the Options field. This column can have one of the following values: **Y** Sets the MQGMO_CONVERT option in the Options field in the MQGMO. This value is the default. **N** Specifies that no data is to be converted.
RCV_ACCEPT_TRUNC_MSG	This column specifies the behavior of the MQ function when oversized messages are retrieved. This column corresponds to the Options field in the get message options structure (MQGMO). MQ functions use the value in this column to set the Options field. This column can have one of the following values: **Y** Sets the MQGMO_ACCEPT_TRUNCATED_MSG option in the Options field in the MQGMO. This value is the default. **N** Specifies that no messages are to be truncated. If the message is too large to fit in the buffer, the MQ function terminates with an error. **Recommendation:** Set this column to Y. In this case, if the message buffer is too small to hold the complete message, the MQ function can fill the buffer with as much of the message as the buffer can hold.

Table 129. SYSIBM.MQPOLICY_TABLE column descriptions (continued)	
Column name	**Description**
REV_OPEN_SHARED	This column specifies the input queue mode when messages are retrieved. This column corresponds to the Options parameter for an MQOPEN call. MQ functions use the value in this column to set the Options parameter. This column can have one of the following values: **S** Sets the MQOO_INPUT_SHARED option. This value is the default. **E** Sets the MQ option MQOO_INPUT_EXCLUSIVE option. **D** Sets the MQ option MQOO_INPUT_AS_Q_DEF option.
SYNCPOINT	This column indicates whether the MQ function is to operate within the protocol for a normal unit of work. This column can have one of the following values: **Y** Specifies that the MQ function is to operate within the protocol for a normal unit of work. Use this value for two-phase commit environments. This value is the default. **N** Specifies that the MQ function is to operate outside the protocol for a normal unit of work. Use this value for one-phase commit environments.
DESC	This column contains the description of the policy.

Related reference

Core WLM environments for Db2-supplied routines (Db2 Installation and Migration)
Developing applications reference

Basic messaging with IBM MQ

The most basic form of messaging with the Db2 MQ functions occurs when all database applications connect to the same Db2 database server. Clients can be local to the database server or distributed in a network environment.

In a simple scenario, client A invokes the MQSEND function to send a user-defined string to the location that is defined by the default service. Db2 executes the MQ functions that perform this operation on the database server. At some later time, client B invokes the MQRECEIVE function to remove the message at the head of the queue that is defined by the default service, and return it to the client. Db2 executes the MQ functions that perform this operation on the database server.

Database clients can use simple messaging in a number of ways:

- Data collection

Information is received in the form of messages from one or more sources. An information source can be any application. The data is received from queues and stored in database tables for additional processing.

- Workload distribution

  Work requests are posted to a queue that is shared by multiple instances of the same application. When an application instance is ready to perform some work, it receives a message that contains a work request from the head of the queue. Multiple instances of the application can share the workload that is represented by a single queue of pooled requests.

- Application signaling

  In a situation where several processes collaborate, messages are often used to coordinate their efforts. These messages might contain commands or requests for work that is to be performed. For more information about this technique, see "Application to application connectivity with IBM MQ" on page 797.

The following scenario extends basic messaging to incorporate remote messaging. Assume that machine A sends a message to machine B.

1. The Db2 client executes an MQSEND function call, specifying a target service that has been defined to be a remote queue on machine B.
2. The MQ functions perform the work to send the message. The WebSphere MQ server on machine A accepts the message and guarantees that it will deliver it to the destination that is defined by the service and the current MQ configuration of machine A. The server determines that the destination is a queue on machine B. The server then attempts to deliver the message to the WebSphere MQ server on machine B, trying again as needed.
3. The IBM MQ server on machine B accepts the message from the server on machine A and places it in the destination queue on machine B.
4. A IBM MQ client on machine B requests the message at the head of the queue.

Sending messages with IBM MQ

When you send messages with IBM MQ, you choose what data to send, where to send it and when to send it. This type of messaging is called *send and forget*; the sender sends a message and relies on IBM MQ to ensure that the message reaches its destination.

Procedure

To send messages with IBM MQ, use MQSEND.

Message content can be any combination of SQL statements, expressions, functions, and user-specified data. Because this MQSEND function uses two-phase commit, the COMMIT statement ensures that the message is added to the MQ queue.

Examples

If you send more than one column of information, separate the columns with the characters `|| ' ' ||`.

`MQSEND (LASTNAME || ' ' || FIRSTNAME)`

The following examples use the DB2MQ schema for two-phase commit, with the default service Db2.DEFAULT.SERVICE and the default policy Db2.DEFAULT.POLICY.

The following SQL SELECT statement sends a message that consists of the string "Testing msg":

```
SELECT DB2MQ.MQSEND ('Testing msg')
  FROM SYSIBM.SYSDUMMY1;
COMMIT;
```

The MQSEND function is invoked once because SYSIBM.SYSDUMMY1 has only one row. Because this MQSEND function uses two-phase commit, the COMMIT statement ensures that the message is added to the queue.

When you use single-phase commit, you do not need to use a COMMIT statement. For example:

```
SELECT DB2MQ.MQSEND ('Testing msg')
  FROM SYSIBM.SYSDUMMY1;
```

The MQ operation causes the message to be added to the queue.

Assume that you have an EMPLOYEE table, with VARCHAR columns LASTNAME, FIRSTNAME, and DEPARTMENT. To send a message that contains this information for each employee in DEPARTMENT 5LGA, issue the following SQL SELECT statement:

```
SELECT DB2MQ.MQSEND (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT)
  FROM EMPLOYEE WHERE DEPARTMENT = '5lGA';
COMMIT;
```

Related reference

MQSEND (Db2 SQL)

Retrieving messages with IBM MQ

With IBM MQ, programs can read or receive messages. Both reading and receiving operations return the message at the start of the queue. However, the reading operation does not remove the message from the queue, whereas the receiving operation does.

About this task

A message that is retrieved using a receive operation can be retrieved only once, whereas a message that is retrieved using a read operation allows the same message to be retrieved many times.

Examples

The following examples use the DB2MQ2N schema for two-phase commit, with the default service Db2.DEFAULT.SERVICE and the default policy Db2.DEFAULT.POLICY.

Example

The following SQL SELECT statement reads the message at the head of the queue that is specified by the default service and policy:

```
SELECT DB2MQ2N.MQREAD()
  FROM SYSIBM.SYSDUMMY1;
```

The MQREAD function is invoked once because SYSIBM.SYSDUMMY1 has only one row. The SELECT statement returns a VARCHAR(4000) string. If no messages are available to be read, a null value is returned. Because MQREAD does not change the queue, you do not need to use a COMMIT statement.

Example

The following SQL SELECT statement causes the contents of a queue to be materialized as a Db2 table:

```
SELECT T.*
  FROM TABLE(DB2MQ2N.MQREADALL()) T;
```

The result table T of the table function consists of all the messages in the queue, which is defined by the default service, and the metadata about those messages. The first column of the materialized result table is the message itself, and the remaining columns contain the metadata. The SELECT statement returns both the messages and the metadata.To return only the messages, issue the following statement:

```
SELECT T.MSG
  FROM TABLE(DB2MQ2N.MQREADALL()) T;
```

The result table T of the table function consists of all the messages in the queue, which is defined by the default service, and the metadata about those messages. This SELECT statement returns only the messages.

Example

The following SQL SELECT statement receives (removes) the message at the head of the queue:

```
SELECT DB2MQ2N.MQRECEIVE()
  FROM SYSIBM.SYSDUMMY1;
COMMIT;
```

The MQRECEIVE function is invoked once because SYSIBM.SYSDUMMY1 has only one row. The SELECT statement returns a VARCHAR(4000) string. Because this MQRECEIVE function uses two-phase commit, the COMMIT statement ensures that the message is removed from the queue. If no messages are available to be retrieved, a null value is returned, and the queue does not change.

Example

Assume that you have a MESSAGES table with a single VARCHAR(2000) column. The following SQL INSERT statement inserts all of the messages from the default service queue into the MESSAGES table in your Db2 database:

```
INSERT INTO MESSAGES
  SELECT T.MSG
    FROM TABLE(DB2MQ2N.MQRECEIVEALL()) T;
COMMIT;
```

The result table T of the table function consists of all the messages in the default service queue and the metadata about those messages. The SELECT statement returns only the messages. The INSERT statement stores the messages into a table in your database.

Application to application connectivity with IBM MQ

Application-to-application connectivity is typically used when putting together a diverse set of application subsystems. To facilitate application integration, WebSphere MQ provides the means to interconnect applications.

The *Request-and-reply* method is very common when interconnecting applications.

Request-and-reply communication method

The request-and-reply method enables one application to request the services of another application. One way to do this is for the requester to send a message to the service provider to request that some work be performed. When the work has been completed, the provider might decide to send results, or just a confirmation of completion, back to the requester. Unless the requester waits for a reply before continuing, IBM MQ must provide a way to associate the reply with its request.

IBM MQ provides a correlation identifier to correlate messages in an exchange between a requester and a provider. The requester marks a message with a known correlation identifier. The provider marks its reply with the same correlation identifier. To retrieve the associated reply, the requester provides that correlation identifier when receiving messages from the queue. The first message with a matching correlation identifier is returned to the requester.

The following examples use the DB2MQ schema for single-phase commit.

Examples of request-and-reply communication

Example

The following SQL SELECT statement sends a message consisting of the string "`Msg with corr id`" to the service MYSERVICE, using the policy MYPOLICY with correlation identifier CORRID1:

```
SELECT DB2MQ.MQSEND ('MYSERVICE', 'MYPOLICY', 'Msg with corr id', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
```

The MQSEND function is invoked once because SYSIBM.SYSDUMMY1 has only one row. Because this MQSEND uses single-phase commit, IBM MQ adds the message to the queue, and you do not need to use a COMMIT statement.

Example

The following SQL SELECT statement receives the first message that matches the identifier CORRID1 from the queue that is specified by the service MYSERVICE, using the policy MYPOLICY:

```
SELECT DB2MQ.MQRECEIVE ('MYSERVICE', 'MYPOLICY', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
```

The SELECT statement returns a VARCHAR(4000) string. If no messages are available with this correlation identifier, a null value is returned, and the queue does not change.

Asynchronous messaging in Db2 for z/OS

Programs can communicate with each other by sending data in messages rather than using constructs like synchronous remote procedure calls. With asynchronous messaging, the program that sends the message proceeds with its processing after sending the message, without waiting for a reply.

If the program needs information from the reply, the program suspends processing and waits for a reply message. If the messaging programs use an intermediate queue that holds messages, the requester program and the receiver program do not need to be running at the same time. The requester program places a request message on a queue and then exits. The receiver program retrieves the request from the queue and processes the request.

Asynchronous operations require that the service provider is capable of accepting requests from clients without notice. An asynchronous listener is a program that monitors message transporters, such as WebSphere MQ, and performs actions based on the message type. An asynchronous listener can use WebSphere MQ to receive all messages that are sent to an endpoint. An asynchronous listener can also register a subscription with a publish or subscribe infrastructure to restrict the messages that are received to messages that satisfy specified constraints.

Examples: The following examples show some common uses of asynchronous messaging:

Message accumulator

You can accumulate the messages that are sent asynchronously so that the listener checks for messages and stores those messages automatically in a database. This database, which acts as a message accumulator, can save all messages for a particular endpoint, such as an audit trail. The asynchronous listener can subscribe to a subset of messages, such as *save only high value stock trades*. The message accumulator stores entire messages, and does not provide for selection, transformation, or mapping of message contents to database structures. The message accumulator does not reply to messages.

Message event handler

The asynchronous event handler listens for messages and invokes the appropriate handler (such as a stored procedure) for the message endpoint. You can call any arbitrary stored procedure. The asynchronous listener lets you select, map, or reformat message contents for insertion into one or more database structures.

Asynchronous messaging has the following benefits:

- The client and database do not need to be available at the same time. If the client is available intermittently, or if the client fails between the time the request is issued and the response is sent, it is still possible for the client to receive the reply. Or, if the client is on a mobile computer and becomes disconnected from the database, and if a response is sent, the client can still receive the reply.

- The content of the messages in the database contain information about when to process particular requests. The messages in the database use priorities and the request contents to determine how to schedule the requests.
- An asynchronous message listener can delegate a request to a different node. It can forward the request to a second computer to complete the processing. When the request is complete, the second computer returns a response directly to the endpoint that is specified in the message.
- An asynchronous listener can respond to a message from a supplied client, or from a user-defined application. The number of environments that can act as a database client is greatly expanded. Clients such as factory automation equipment, pervasive devices, or embedded controllers can communicate with Db2 either directly through IBM MQ or through some gateway that supports WebSphere MQ.

MQListener in Db2 for z/OS

Db2 for z/OS provides an asynchronous listener, MQListener. MQListener is a framework for tasks that read from IBM MQ queues and call Db2 stored procedures with messages as those messages arrive.

MQListener combines messaging with database operations. You can configure the MQListener daemon to listen to the IBM MQ message queues that you specify in a configuration database. MQListener reads the messages that arrive from the queue and calls Db2 stored procedures using the messages as input parameters. If the message requires a reply, MQListener creates a reply from the output that is generated by the stored procedure. The message retrieval order is fixed at the highest priority first, and then within each priority the first message received is the first message served.

MQListener runs as a single multi-threaded process on z/OS UNIX System Services. Each thread or task establishes a connection to its configured message queue for input. Each task also connects to a Db2 database on which to run the stored procedure. The information about the queue and the stored procedure is stored in a table in the configuration database. The combination of the queue and the stored procedure is a task.

MQListener tasks are grouped together into named configurations. By default, the configuration name is empty. If you do not specify the name of a configuration for a task, MQListener uses the configuration with an empty name.

Transaction support

There is support for both one-phase and two-phase commit environments. A one-phase commit environment is where DB interactions and MQ interactions are independent. A two-phase commit environment is where DB interactions and MQ interactions are combined in a single unit of work.

'db2mqln1' is the name of the executable for one phase and 'db2mqln2' is the name of the executable for two phase.

Logical ordering of messages

The two-phase commit version of the MQListener stored procedure processes messages that are in a group in logical order. The single-phase commit version of the MQListener stored procedure processes messages that are in a group in physical order.

Stored procedure interfaces

There are two stored procedure interfaces support MQListener:

Stored procedure interface with two parameters

This stored procedure interface for MQListener takes the incoming message as input and returns the reply, which might be NULL, as output. For example:

```
CREATE schema.proc(
 IN INMSG inMsgType,
 OUT OUTMSG outMsgType)...
```

The data type for INMSG and the data type for OUTMSG can be VARCHAR, VARBINARY, CLOB, or BLOB, of any length, and are determined at startup. The input data type and output data type can be different data types. If an incoming message is a request and has a specified reply-to queue, the message in OUTMSG is sent to the specified queue. The incoming message can be one of the following message types:

- Datagram
- Datagram with report requested
- Request message with reply
- Request message with reply and report requested

Stored procedure interface with three parameters

This stored procedure interface for MQListener has parameters with the following information:

- An incoming message as input
- A reply, which might be NULL, as output
- A message header, which can be input or output

For example:

```
CREATE schema.proc(
 IN INMSG inMsgType,
 OUT OUTMSG outMsgType,
 INOUT MSGHEADER msgHeaderType)...
```

The data type for INMSG and the data type for OUTMSG can be VARCHAR, VARBINARY, CLOB, or BLOB, of any length, and are determined at startup. The input data type and output data type can be different data types. If an incoming message is a request and has a specified reply-to queue, the message in OUTMSG is sent to the specified queue. The incoming message can be one of the following message types:

- Datagram
- Datagram with report requested
- Request message with reply
- Request message with reply and report requested

The data type for MSGHEADER can be VARBINARY or BLOB. The minimum length of MSGHEADER is 324, which is the size of the message queuing message descriptor (MQMD) structure for IBM MQ messages.

MQListener passes the message header to the stored procedure in the MSGHEADER parameter. The stored procedure can get the message descriptor properties from the MSGHEADER parameter. If the message is a request message, the stored procedure can specify the properties for the reply queue and reply queue manager in the MSGHEADER parameter. The output message in OUTMSG is sent to that specified queue.

Configuring MQListener in Db2 for z/OS

Before you can use MQListener, you must configure your database environment so that your applications can use messaging with database operations. You must also configure IBM MQ for MQListener.

Before you begin

Ensure that the person who runs the installation job has required authority to create the configuration table and to bind the DBRMs.

About this task

The following sample jobs for MQListener are located in the *prefix*.SDSNSAMP data set.

DSNTIJML
: A sample job for extracting library files and configuring MQListener in z/OS UNIX System Services

DSNTEJML
: A sample job that runs scripts for configuring MQListener.

DSNTEJSP
: A sample job for extracting library tar files in z/OS UNIX System Services when applying PTFs for MQListener.

Procedure

To configure the environment for MQListener and to develop a simple application that receives a message, inserts the message in a table, and creates a simple response message, use these steps:

1. Configure MQListener to run in the Db2 environment, so that your applications can use messaging with database operations, by completing the following steps:
   a) In z/OS UNIX System Services, the default path for MQListener in is `/usr/lpp/db2b10/mql`. The `mqlsn.tar.Z` tar file forMQListener is located in this path. If MQListener is not installed in the default path, replace all occurrences of the default path in the samples DSNTEJML, DSNTEJSP and DSNTIJML with the path name where MQListener is installed before you run DSNTIJML.
   b) Customize and run installation job DSNTIJML. It completes the following actions:
      i) Extracts the necessary files and libraries to z/OS UNIX System Services under the path where MQListener is installed.
      ii) Creates the MQListener configuration table (SYSMQL.LISTENERS) in the default database DSNDB04.
      iii) Binds the DBRMs to the plan DB2MQLSN.
   c) Follow the instructions in the README file in the MQListener installation path to complete the configuration process.

   Applying PTFs:

   When applying PTFs for MQListener, run job DSNTJESP to extract the tar file for the MQListener library files in z/OS UNIX System Services.
2. Configure IBM MQ for MQListener.

   You can run a simple MQListener application with a simple IBM MQ configuration. More complex applications might need a more complex configuration. Configure at least two kinds of IBM MQ entities: the queue manager and some local queues. Configure these entities for use in such instances as transaction management, deadletter queue, backout queue, and backout retry threshold.
   a) Create IBM MQ QueueManager. Define the IBM MQ subsystem to z/OS and then issue the following command from a z/OS console to start the queue manager, where *command-prefix* is the command prefix for the IBM MQ subsystem.

   ```
   command-prefix START QMGR
   ```

   b) Create Queues under IBM MQ QueueManager:

For example, n a simple MQListener application, you typically use the following IBM MQ queues:

Deadletter queue
: The deadletter queue in IBM MQ holds messages that cannot be processed. MQListener uses this queue to hold replies that cannot be delivered, for example, because the queue to which the replies should be sent is full. A deadletter queue is useful in any MQ installation especially for recovering messages that are not sent.

Backout queue
: For MQListener tasks that use two-phase commit, the backout queue serves a similar purpose as the deadletter queue. MQListener places the original request in the backout queue after the request is rolled back a specified number of times (called the backout threshold).

Administration queue
: The administration queue is used for routing control messages, such as shutdown and restart, to MQListener. If you do not supply an administration queue, the only way to shut down MQListener is to issue a kill command.

Application input and output queues
: The application uses input queues and output queues. The application receives messages from the input queue and sends replies and exceptions to the output queue.

Create your local queues by using CSQUTIL utility or by using IBM MQ operations and control panels from ISPF (csqorexx). The following is an example of the JCL that is used to create your local queues. In this example, MQND is the name of the queue manager:

```
//*
//*  ADMIN_Q      : Admin queue
//*  BACKOUT_Q    : Backout queue
//*  IN_Q         : Input queue having a backout queue with threshold=3
//*  REPLY_Q      : output queue or reply queue
//*  DEADLLETTER_Q: Dead letter queue
//*
//DSNTECU   EXEC PGM=CSQUTIL,PARM='MQND'
//STEPLIB   DD   DSN=MQS.SCSQANLE,DISP=SHR
//          DD   DSN=MQS.SCSQAUTH,DISP=SHR
//SYSPRINT  DD   SYSOUT=*
//SYSIN     DD *
  COMMAND     DDNAME(CREATEQ)
/*
//CREATEQ DD *
     DEFINE QLOCAL('ADMIN_Q') REPLACE +
            DESCR('INPUT-OUTPUT')   +
            PUT(ENABLED)       +
            DEFPRTY(0)         +
            DEFPSIST(NO)       +
            SHARE              +
            DEFSOPT(SHARED)    +
            GET(ENABLED)
     DEFINE QLOCAL('BACKOUT_Q') REPLACE +
            DESCR('INPUT-OUTPUT')   +
            PUT(ENABLED)       +
            DEFPRTY(0)         +
            DEFPSIST(NO)       +
            SHARE              +
            DEFSOPT(SHARED)    +
            GET(ENABLED)
     DEFINE QLOCAL('REPLY_Q') REPLACE +
            DESCR('INPUT-OUTPUT')   +
            PUT(ENABLED)       +
            DEFPRTY(0)         +
            DEFPSIST(NO)       +
            SHARE              +
            DEFSOPT(SHARED)    +
            GET(ENABLED)
      DEFINE QLOCAL('IN_Q') REPLACE +
            DESCR('INPUT-OUTPUT')   +
            PUT(ENABLED)       +
            DEFPRTY(0)         +
            DEFPSIST(NO)       +
            SHARE              +
            DEFSOPT(SHARED)    +
            GET(ENABLED)       +
            BOQNAME('BACKOUT_Q') +
            BOTHRESH(3)
```

```
      DEFINE QLOCAL('DEADLETTER_Q') REPLACE +
            DESCR('INPUT-OUTPUT')   +
            PUT(ENABLED)       +
            DEFPRTY(0)         +
            DEFPSIST(NO)       +
            SHARE              +
            DEFSOPT(SHARED)    +
            GET(ENABLED)
      ALTER QMGR DEADQ ('DEADLETTER_Q') REPLACE
 /*
```

3. Configure MQListener tasks. For more information, see "Configuring MQListener tasks" on page 804.
4. Create a stored procedure that MQListener uses to store messages in a table. See "Creating a sample stored procedure to use with MQListener" on page 807 for details.
5. Run a simple MQListener application.

Environment variables for logging and tracing MQListener
Several environment variables control logging and tracing for MQListener. These variables are defined in the file .profile.

MQLSNTRC
When this environment variable is set to 1, MQListener writes function entry, data, and exit points to a unique HFS or zFS file. A unique trace file is generated whenever any of the MQListener commands are run. This trace file is used by IBM Support for debugging. Do not define this variable unless IBM Support requests that you do so.

If you have enabled tracing, when the MQListener daemon is running, it writes to the trace file. Therefore, if you open the trace file while the MQListener daemon is running, you need to open it only in read mode.

MQLSNLOG
The log file contains diagnostic information about major events. This environment variable is set to the name of the file where all log information is written. All instances of the MQListener daemon running one or more tasks share the same file. For monitoring the MQListener daemon, this variable should always be set.

When the MQListener daemon is running, it writes to the log file. Therefore, if you open the log file while the MQListener daemon is running, you need to open it only in read mode.

MQLSNLWR
When MQLSNLOG specifies an HFS log file, MQLSNLWR provides the capability to limit the HFS log file size. The syntax for an MQLSNLWR export command is:

```
export MQLSNLWR=file-size,file-name
```

The meanings of the variables are:

file-size
The maximum size of the MQListener log file, in megabytes.

file-name
The name of the HFS file into which MQListener saves a copy of the MQListener log file.

When MQLSNLWR is specified, and the MQListener HFS log file reaches *file-size*, MQListener saves a copy of the log file named *file-name*, and reinitializes the HFS log file.

Important: The ID under which MQListener runs must have write access to the HFS log file and to the copy of the HFS log file that is specified by *file-name*. If MQListener cannot open or write to the copy of the HFS log file, MQListener reinitializes the HFS log file, but does not create a copy.

Refer to the README file for more details about these variables.

Configuration table: SYSMQL.LISTENERS
If you use MQListener, you must create the MQListener configuration table SYSMQL.LISTENERS by running installation job DSNTIJML.

The SYSMQL.LISTENERS table contains a row for each configuration that you create when you issue MQListener db2mqln1 or db2mqln2 configuration commands.

The following table describes each of the columns of the configuration table SYSMQL.LISTENERS.

Table 130. Description of columns in the SYSMQL.LISTENERS table

Column name	Description
CONFIGURATIONNAME	The configuration name. The configuration name enables you to group several tasks into the same configuration. A single instance of MQListener can run all of the tasks that are defined within a configuration name.
QUEUEMANAGER	The name of the IBM MQ subsystem that contains the queues that are to be used.
INPUTQUEUE	The name of the queue in the WebSphere MQ subsystem that is to be monitored for incoming messages. The combination of the input queue and the queue manager are unique within a configuration
PROCNODE	Currently unused
PROCSCHEMA	The schema name of the stored procedure that will be called by MQListener
PROCNAME	The name of the stored procedure that will be called by MQListener
PROCTYPE	Currently unused
NUMINSTANCES	The number of duplicate instances of a single task that are to run in this configuration
WAITMILLIS	The time MQListener waits (in milliseconds) after processing the current message before it looks for the next message
MINQUEUEDEPTH	Currently unused

Configuring MQListener tasks

As part of configuring MQListener in Db2 for z/OS, you must configure at least one MQListener task.

About this task

Use MQListener command **`db2mqln1`** or **`db2mqln2`** to configure MQListener tasks. Issue the command from the z/OS UNIX System Services command line in any directory. Alternatively, you can put the command in a file, grant execute permission on the file, and use the BPXBATCH utility to invoke the command using JCL. Sample script files are provided and are located in the `/`*`mqlistener-install-path`*`/mqlsn/listener/script` directory in z/OS UNIX System Services. Sample JCL is also provided in member DSNTEJML of data set *prefix*.SDSNSAMP. When you run MQListener commands, configuration information is stored in the Db2 table SYSMQL.LISTENERS.

The command parameters are:

-adminQueue
: The queue to which MQListener listens for administration commands. If **`-adminQueue`** is not specified, applications do not receive any administration commands through the message queue.

-adminQMgr
The name of the IBM MQ subsystem that contains the queues that are to be used for administrative tasks. If **-adminQMgr** is not specified, the configured default queue manager is used.

-config
A name that identifies a group of tasks that run together. If -config is not specified, the default configuration is run.

-queueManager
The name of the IBM MQ subsystem that contains the queues that are to be used. If **-queueManager** is not specified, is not specified, the default queue manager is used.

-inputQueue
The name of the queue in the IBM MQ subsystem that is to be monitored for incoming messages. The combination of the -inputQueue value and the **-queueManager** value must be unique within a configuration.

-numInstances
The number of duplicate instances of a single task that are to run in a configuration.

-numMessagesCommit
The number of messages that are received before MQListener issues a COMMIT. The default is 1. This option is supported only for **db2mqln2**.

-procName
The name of the stored procedure that MQListener calls when it detects that a message is received.

-procSchema
The schema name of the stored procedure that MQListener calls when it detects that a message is received.

-ssID
The subsystem where the MQListener daemon runs. Configuration information is stored in this subsystem.

-timeRestart
If a stored procedure that is specified by **-procSchema** and **-procName** fails at MQListener startup time, the number of seconds that threads that are running with that stored procedure suspend before repeating the setup process. MQListener continues startup for threads that do not use that stored procedure. This value must be an integer between 0 and 7200. 0 is the default.

-restartDB2
Whether MQListener automatically reconnects and resumes processing after Db2 is stopped and restarted.

'Y'
MQListener automatically reconnects and resumes processing after Db2 is stopped and restarted.

'N'
MQListener does not automatically reconnect after Db2 is stopped and restarted. 'N' is the default value.

The syntax of the commands follows. In the command syntax, *mqlistener-command* is `db2mqln1` or `db2mqln2`.

- To add an MQListener configuration, issue the *mqlistener-command* **add** command:

```
mqlistener-command add
   -ssID subsystem-name
   -config configuration-name
   -queueManager queuemanager-name
   -inputQueue inputqueue-name
   -procName stored-procedure-name
   -procSchema stored-procedure-schema name
   -numInstances number-of-instances
```

- To display information about the configuration, issue the following *mqlistener-command* **show** command:

```
mqlistener-command show
   -ssID subsystem-name
   -config configuration-name
```

To display information about all the configurations, issue the *mqlistener-command* **show** command:

```
mqlistener-command show
   -ssID subsystem-name
   -config all
```

- To remove the messaging tasks, issue the *mqlistener-command* **remove** command:

```
mqlistener-command remove
   -ssID subsystem-name
   -config configuration-name
   -queueManager queuemanager-name
   -inputQueue inputqueue-name
```

- To run the MQListener task, issue the *mqlistener-command* **run** command:

```
mqlistener-command run
   -ssID subsystem-name
   -config configuration-name
   -adminQueue adminqueue-name
   -adminQMgr adminqueuemanager-name
   -numMessagesCommit number-of-messages-before-commit
   -timeRestart number-of-seconds-to-suspend-before-restart
   -restartDB2 y-or-n
```

- To shutdown the MQListener daemon, issue the *mqlistener-command* **admin** command:

```
mqlistener-command admin
   -adminQueue adminqueue-name
   -adminQMgr adminqueuemanager-name
   -adminCommand shutdown
```

- To restart the MQListener daemon, issue the following command:

```
mqlistener-command mqlistener-command admin
    -adminQueue adminqueue-name
    -adminQMgr adminqueuemanager-name
    -adminCommand restart
```

- To get help with the command and the valid parameters, issue the *mqlistener-command* **help** command:

```
mqlistener-command help
```

- To get help for a particular parameter, issue the *mqlistener-command* **help** command, where *command* is a specific parameter:

```
mqlistener-command help command
```

Restriction:

- Use the same queue manager for the request queue and the reply queue.
- MQListener does not support logical messages that are composed of multiple physical messages. MQListener processes physical messages independently.

Creating a sample stored procedure to use with MQListener

You can create a sample stored procedure, APROC, that can be used by MQListener to store a message in a table. The stored procedure returns the string OK if the message is successfully inserted into the table.

Procedure

The following steps create Db2 objects that you can use with MQListener applications:

1. Create a table using SPUFI, DSNTEP2, or the command line processor in the subsystem where you want to run MQListener:

   ```
   CREATE TABLE PROCTABLE (MSG VARCHAR(25) CHECK (MSG NOT LIKE 'FAIL%'));
   ```

   The table contains a check constraint so that messages that start with the characters `FAIL` cannot be inserted into the table. The check constraint is used to demonstrate the behavior of MQListener when the stored procedure fails.
2. Create the following SQL procedure, and define it to the same Db2 subsystem.

   ```
   CREATE PROCEDURE TEST.APROC (
      IN PIN VARCHAR(25),
      OUT POUT VARCHAR(2),
      INOUT PMSGHEADER VARBINARY(500))
     VERSION V1
     LANGUAGE SQL
     PROCEDURE1: BEGIN
      DECLARE REPLYQ  VARBINARY(48);
      DECLARE REPLYQM VARBINARY(48);
      SET REPLYQ = VARBINARY(CONCAT('NEWREPLYQUEUE',X'00'));
      SET REPLYQM = VARBINARY(CONCAT('CSQ1',X'00'));
      SET PMSGHEADER = INSERT(PMSGHEADER,101,LENGTH(REPLYQ),REPLYQ);
      SET PMSGHEADER = INSERT(PMSGHEADER,149,LENGTH(REPLYQM),REPLYQM);
      INSERT INTO SYSADM.PROCTABLE VALUES (PIN);
      SET POUT = 'OK';
     END PROCEDURE1
   ```
3. Add the following configuration, named ACFG, to the configuration table by issuing this command:

   ```
   db2mqln2 add
      -ssID DB2A
      -config ACFG
      -queueManager CSQ1
      -inputQueue DB2MQ_DEFAULT_Q
      -procName APROC
      -procSchema TEST
   ```
4. Run the MQListener daemon for two-phase commit for configuration ACFG. To run MQListener with all of the tasks that are specified in the configuration, issue the following command:

   ```
   db2mqln2 run
      -ssID DB2A
      -config ACFG
      -adminQueue ADMIN_Q
      -adminQMgr MQND
   ```
5. Send a request to the input queue, 'DB2MQ_DEFAULT_Q ', with the message 'another sample message'.
6. Query table PROCTABLE to verify that the sample message was inserted:

   ```
   SELECT * FROM PROCTABLE;
   ```
7. Display the number of messages that remain on the input queue, to verify that the message has been removed. To do that issue the following command from a z/OS console:

   ```
   /-CSQ1 display queue('DB2MQ_DEFAULT_Q ') curdepth
   ```
8. Look at the ReplytoQ name that you specified, to verify that the string 'OK' is generated by the stored procedure.

MQListener error processing

MQListener reads from IBM MQ message queues and calls Db2 stored procedures with those messages. If any errors occur during this process and the message is to be sent to the deadletter queue, MQListener returns a reason code to the deadletter queue.

Specifically, MQListener performs the following actions:

- prefixes the message with an MQ dead letter header (MQDLH) structure
- sets the reason field in the MQDLH structure to the appropriate reason code
- sends the message to the deadletter queue

The following table describes the reason codes that the MQListener daemon returns.

Table 131. Reason codes that MQListener returns

Reason code	Explanation
900	The call to a stored procedure was successful but an error occurred during the Db2 commit process and either of the following conditions were true: • No exception report was requested.[1] • An exception report was requested, but could not be delivered. This reason code applies only to one-phase commit environments.
901	The call to the specified stored procedure failed and the disposition of the MQ message is that an exception report be generated and the original message be sent the deadletter queue.
902	All of the following conditions occurred: • The disposition of the MQ message is that an exception report is not to be generated.[1] • The stored procedure was called unsuccessfully the number of times that is specified as the backout threshold. • The name of the backout queue is the same as the deadletter queue. This reason code applies only to two-phase commit environments.
MQRC_TRUNCATED_ MSG__FAILED	The size of the MQ message is greater than the input parameter of the stored procedure that is to be invoked. In one-phase commit environments, this oversized message is sent to the dead letter queue. In two-phase commit environments, this oversized message is sent to the deadletter queue only when the message cannot be delivered to the backout queue.

Note:

1. To specify that the receiver application generate exception reports if errors occur, set the report field in the MQMD structure that was used when sending the message to one of the following values:
   - MQRO_EXCEPTION
   - MQRO_EXCEPTION_WITH_DATA
   - MQRO_EXCEPTION_WITH_FULL_DATA

Related reference

IBM MQ home

MQListener examples

The application receives a message, inserts the message into a table, and generates a simple response message.

To simulate a processing failure, the application includes a check constraint on the table that contains the message. The constraint prevents any string that begins with the characters 'fail' from being inserted into the table. If you attempt to insert a message that violates the check constraint, the example application returns an error message and re-queues the failing message to the backout queue.

In this example, the following assumptions are made:

- MQListener is installed and configured for subsystem DB7A.
- MQND is the name of IBM MQ subsystem that is defined. The Queue Manager is running, and the following local queues are defined in the DB7A subsystem:

  ADMIN_Q : Admin queue
  BACKOUT_Q : Backout queue
  IN_Q : Input queue that has a backout queue withthreshold = 3
  REPLY_Q : Output queue or Reply queue
  DEADLLETTER_Q : Dead letter queue
- The person who is running the MQListener daemon has execute permission on the DB2MQLSN plan.

Before you run the MQListener daemon, add the following configuration, named ACFG, to the configuration table by issuing the following command:

```
db2mqln2 add
   -ssID DB7A
   -config ACFG
   -queueManager MQND
   -inputQueue IN_Q
   -procName APROC
   -procSchema TEST
```

Run the MQListener daemon for two-phase commit for configuration ACFG by issuing the following command:

```
db2mqln2 run
   -ssID DB7A
   -config ACFG
   -adminQueue ADMIN_Q
   -adminQMgr MQND
   -numMessagesCommit 1
   -timeRestart 60
```

The following examples show how to use MQListener to send a simple message and then inspect the results of the message in the IBM MQ queue manager and the database. The examples include queries to determine if the input queue contains a message or to determine if a record is placed in the table by the stored procedure.

MQListener example 1: Running a simple application:

1. Start with a clean database table by issuing the following SQL statement:

   ```
   delete from PROCTABLE
   ```

2. Send a datagram to the input queue, 'IN_Q', with the message as 'sample message'. Refer to WebSphere MQ sample CSQ4BCK1 to send a message to the queue. Specify the MsgType option for 'Message Descriptor' as 'MQMT_DATAGRAM'.
3. Query the table by using the following statement to verify that the sample message is inserted:

   ```
   select * from PROCTABLE
   ```

4. Display the number of messages that remain on the input queue to verify that the message has been removed. Issue the following command from a z/OS console:

```
/-MQND display queue('In_Q') curdepth
```

MQListener example 2: Sending requests to the input queue and inspecting the reply:

1. Start with a clean database table by issuing the following SQL statement:

```
delete from PROCTABLE
```

2. Send a request to the input queue, 'IN_Q', with the message as 'another sample message'. Refer to IBM MQ sample CSQ4BCK1 to send a message to the queue. Specify the MsgType option for 'Message Descriptor' as 'MQMT_REQUEST' and the queue name for ReplytoQ option.
3. Query the table by using the following statement to verify that the sample message is inserted:

```
select * from PROCTABLE
```

4. Display the number of messages that remain on the input queue to verify that the message has been removed. Issue the following command from a z/OS console:

```
/-MQND display queue('In_Q') curdepth
```

5. Look at the ReplytoQ name that you specified when you sent the request message for the reply by using the IBM MQ sample program CSQ4BCJ1. Verify that the string 'OK' is generated by the stored procedure.

MQListener example 3: Testing an unsuccessful insert operation: If you send a message that starts with the string 'fail', the constraint in the table definition is violated, and the stored procedure fails.

1. Start with a clean database table by issuing the following SQL statement:

```
delete from PROCTABLE
```

2. Send a request to the input queue, 'IN_Q', with the message as 'failing sample message'. Refer to IBM MQ sample CSQ4BCK1 to send a message to the queue. Specify the MsgType option for 'Message Descriptor' as 'MQMT_REQUEST' and the queue name for ReplytoQ option.
3. Query the table by using the following statement to verify that the sample message is not inserted:

```
select * from PROCTABLE
```

4. Display the number of messages that remain on the input queue to verify that the message has been removed. Issue the following command from a z/OS console:

```
/-MQND display queue('In_Q') curdepth
```

5. Look at the Backout queue and find the original message by using the WebSphere MQ sample program CSQ4BCJ1.

Note: In this example, if a request message with added options for 'exception report' is sent (the Report option is specified for 'Message Descriptor'), an exception report is sent to the reply queue and the original message is sent to the deadletter queue.

Chapter 7. Db2 as a web services consumer and provider

Web services are a set of resources and components that applications can use over HTTP. You can use Db2 as a web services provider and a web services consumer.

Db2 as a web services consumer

Db2 can act as a client for web services, which enables you to be a consumer of web services in your Db2 applications.

SOAP web services Simple Object Access Protocol (SOAP) is an XML protocol that consists of the following characteristics:

- An envelope that defines a framework for describing the contents of a message and how to process the message
- A set of encoding rules for expressing instances of application-defined data types
- A convention for representing SOAP requests and responses

A set of SOAP functions is provided by Db2 and is installed and configured when you install or migrate Db2.

REST web services The Representational State Transfer (REST) protocol provides access to web-based content directly from SQL statements through HTTP requests. A set of basic sample REST user-defined functions can be installed with Db2. These functions provide access to web-based content through the HTTP GET, POST, PUT, and DELETE methods.

Db2 as a web services provider

You can enable your Db2 data and applications as web services through the Web Services Object Runtime Framework (WORF). You can define a web service in Db2 by using a Document Access Definition Extension (DADX). In the DADX file, you can define web services based on SQL statements and stored procedures. Based on your definitions in the DADX file, WORF performs the following actions:

- Handles the connection to Db2 and the execution of the SQL and the stored procedure call
- Converts the result to a web service
- Handles the generation of any Web Services Definition Language (WSDL) and UDDI (Universal Description, Discovery, and Integration) information that the client application needs

Related concepts

Sample REST user-defined functions (Db2 Installation and Migration)

Deprecated: The SOAPHTTPV and SOAPHTTPC user-defined functions

Db2 provides user-defined functions that allow you to work with SOAP and consume web services in SQL statements. The user-defined functions are two varieties of SOAPHTTPV for VARCHAR data and two varieties of SOAPHTTPC for CLOB data.

Restriction: SOAPHTTPV and SOAPHTTPC user-defined functions have been deprecated. Use SOAPHTTPNV and SOAPHTTPNC user-defined functions instead.

The user-defined functions perform the following actions:

1. Compose a SOAP request
2. Post the request to the service endpoint

3. Receive the SOAP response
4. Return the content of the SOAP body

When a consumer receives the result of a web services request, the SOAP envelope is stripped and the XML document is returned. An application program can process the result data and perform a variety of operations, including inserting or updating a table with the result data.

SOAPHTTPV and SOAPHTTPC are user-defined functions that enable Db2 to work with SOAP and to consume web services in SQL statements. These functions are overloaded functions that are used for VARCHAR or CLOB data of different sizes, depending on the SOAP body. Web services can be invoked in one of four ways, depending on the size of the input data and the result data. SOAPHTTPV returns VARCHAR(32672) data and SOAPHTTPC returns CLOB(1M) data. Both functions accept either VARCHAR(32672) or CLOB(1M) as the input body.

Example: The following example shows an HTTP post header that posts a SOAP request envelope to a host. The SOAP envelope body shows a temperature request for Barcelona.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: services.xmethods.net
Connection: Keep-Alive User-Agent: DB2SOAP/1.0
Content-Type: text/xml; charset="UTF-8"
SOAPAction: ""
Content-Length: 410

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                   xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
                   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                   xmlns:xsd=http://www.w3.org/2001/XMLSchema >
  <SOAP-ENV:Body>
     <ns:getTemp xmlns:ns="urn:xmethods-Temperature">
           <city>Barcelona</city>
     </ns:getTemp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: The following example is the result of the preceding example. This example shows the HTTP response header with the SOAP response envelope. The result shows that the temperature is 85 degrees Fahrenheit in Barcelona.

```
HTTP/1.1 200 OK
Date: Wed, 31 Jul 2002 22:06:41 GMT
Server: Enhydra-MultiServer/3.5.2
Status: 200
Content-Type: text/xml; charset=utf-8
Servlet-Engine: Lutris Enhydra Application Server/3.5.2
   (JSP 1.1; Servlet 2.2; Java 1.3.1_04;
    Linux 2.4.7-10smp i386; java.vendor=Sun Microsystems Inc.)
Content-Length: 467
Set-Cookie:JSESSIONID=JLEcR34rBc2GTIkn-0F51ZDk;Path=/soap
X-Cache: MISS from www.xmethods.net
Keep-Alive: timeout=15, max=10
Connection: Keep-Alive

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                   xmlns:xsd=http://www.w3.org/2001/XMLSchema >
   <SOAP-ENV:Body>
     <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/ >
         <return xsi:type="xsd:float">85</return>
     </ns1:getTempResponse>
   </SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Example: The following example shows how to insert the result from a web service into a table

```
INSERT INTO MYTABLE(XMLCOL) VALUES (DB2XML.SOAPHTTPC(
   'http://www.myserver.com/services/db2sample/list.dadx/SOAP',
   'http://tempuri.org/db2sample/list.dadx'
   '<listDepartments xmlns="http://tempuri.org/db2sample/listdadx">
        <deptno>A00</deptno>
    </ListDepartments>'))
```

The SOAPHTTPNV and SOAPHTTPNC user-defined functions

Db2 provides SOAPHTTPNV and SOAPHTTPNC user-defined functions that allow you to work with SOAP and consume web services in SQL statements. The user-defined functions are two varieties of SOAPHTTPNV for VARCHAR data and two varieties of SOAPHTTPNC for CLOB data.

The user-defined functions perform the following actions:

1. Post the input SOAP request to the service endpoint
2. Receive and return the SOAP response

SOAPHTTPNV and SOAPHTTPNC allow you to specify a complete SOAP message as input and return complete SOAP messages from the specified web service as a CLOB or VARCHAR representation of the returned XML data. . SOAPHTTPNV returns VARCHAR(32672) data and SOAPHTTPNC returns CLOB(1M) data. Both functions accept either VARCHAR(32672) or CLOB(1M) as the input body.

SOAPHTTPNV and SOAPHTTPNC user-defined functions can support SOAP 1.1 or SOAP 1.2. Check with your system administrator to determine which levels of SOAP are supported by the user-defined functions in your environment.

Example

The following example shows how to insert the complete result from a web service into a table using SOAPHTTPNC.

```
INSERT INTO EMPLOYEE(XMLCOL)
     VALUES (DB2XML.SOAPHTTPNC(
        'http://www.myserver.com/services/db2sample/list.dadx/SOAP',
        'http://tempuri.org/db2sample/list.dadx',
        '<?xml version="1.0" encoding="UTF-8" ?>' ||
        '<SOAP-ENV:Envelope ' ||
        'xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ' ||
        'xmlns:xsd="http://www.w3.org/2001/XMLSchema" ' ||
        'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' ||
        '<SOAP-ENV:Body>' ||
        '<listDepartments xmlns="http://tempuri.org/db2sample/list.dadx">
            <deptNo>A00</deptNo>
         </listDepartments>' ||
        '</SOAP-ENV:Body>' ||
        '</SOAP-ENV:Envelope>'))
```

Related tasks

Additional steps for enabling web service user-defined functions (Db2 Installation and Migration)

SQLSTATEs for Db2 as a web services consumer

Db2 returns SQLSTATE values for error conditions that are related to using Db2 as a web services consumer.

The following tables show possible SQLSTATE values.

Table 132. SQLSTATE values for SOAPHTTPV and SOAPHTTPC user-defined functions

SQLSTATE	Description
38301	An unexpected NULL value was pass as input to the function.
38302	The function was unable to allocate space.
38304	An unknown protocol was specified ion the endpoint URL.
38305	An invalid URL was specified on the endpoint URL.
38306	An error occurred while attempting to create a TCP/IP socket.
38307	An error occurred while attempting to bind a TCP/IP socket.

Table 132. SQLSTATE values for SOAPHTTPV and SOAPHTTPC user-defined functions (continued)

SQLSTATE	Description
38308	The function could not resolve the specified host name.
38309	An error occurred while attempting to connect to the specified server.
38310	An error occurred while attempting to retrieve information from the protocol.
38311	An error occurred while attempting to set socket options.
38312	The function received unexpected data returned for the web service.
38313	The web service did not return data of the proper content type.
38314	An error occurred while initializing the XML parser.
38315	An error occurred while creating the XML parser.
38316	An error occurred while establishing a handler for the XML parser.
38317	The XML parser encountered an error while parsing the result data.
38318	The XML parser could not convert the result data to the database code page.
38319	The function could not allocate memory when creating a TCP/IP socket.
38320	An error occurred while attempting to send the request to the specified server.
38321	The function was unable to send the entire request to the specified server.
38322	An error occurred while attempting to read the result data from the specified server.
38323	An error occurred while waiting for data to be returned from the specified server.
38324	The function encountered an internal error while attempting to format the input message.
38325	The function encountered an internal error while attempting to add namespace information to the input message.
38327	The XML parser could not strip the SOAP envelope from the result message.
38328	An error occurred while processing an SSL connection.

Table 133. SQLSTATE values for SOAPHTTPNV and SOAPHTTPNC user-defined functions

SQLSTATE	Description
38350	An unexpected NULL value was specified for the endpoint, action, or SOAP input.
38351	A dynamic memory allocation error.
38352	An unknown or unsupported transport protocol.
38353	An invalid URL was specified.
38354	An error occurred while resolving the hostname.
38355	A memory exception for socket.
38356	An error occurred during socket connect.
38357	An error occurred while setting socket options.
38358	An error occurred during input/output control (ioctl) to verify HTTPS enablement.
38359	An error occurred while reading from the socket.

Table 133. SQLSTATE values for SOAPHTTPNV and SOAPHTTPNC user-defined functions (continued)

SQLSTATE	Description
38360	An error occurred due to socket timeout.
38361	No response from the specified host.
38362	An error occurred due to an unexpected HTTP return or content type
38363	The TCP/IP stack was not enabled for HTTPS.

Related tasks

Additional steps for enabling web service user-defined functions (Db2 Installation and Migration)

Chapter 8. Application compatibility levels in Db2

The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

When your Db2 environment is migrated to new-function mode, you can run applications with the features and behavior of previous versions or Db2 11.

You can change the application compatibility level for each application when you are ready for it to run with the features and behavior of a particular Db2 version.

The application compatibility of a package is initially set when you bind a package, based on the following values:

1. The APPLCOMPAT bind option value, if specified.
2. If the bind option is omitted, the APPLCOMPAT subsystem parameter.

For static SQL statements, the APPLCOMPAT column of the SYSIBM.SYSPACKAGE catalog table stores the application compatibility setting. This setting changes for the following reasons:

- You issue a REBIND command for the package and specify a different value for the APPLCOMPAT option. If you omit this option, the previous value for the package is used. If no previous value is available (such as for packages last bound before the introduction of application compatibility) the APPLCOMPAT subsystem parameter value is used.
- An automatic bind of the package occurs. The application compatibility is set to the previous value. If no previous value is available, the APPLCOMPAT subsystem parameter value is used.

For dynamic SQL statements, the CURRENT APPLICATION COMPATIBILITY special register stores the application compatibility setting. This setting changes for the following reasons:

- The special register is initialized to the application compatibility of the package, as described above.
- During execution of the package, SET CURRENT APPLICATION COMPATIBILITY statements can change the special register. The value must be equivalent to the APPLCOMPAT bind option for the package or lower, if the value is V12R1M500 or above.

The default APPLCOMPAT subsystem parameter value for a new installation is set to V11R1. For a migrated environment, it is set to the previous V10R1.

Tip: When you migrate to Db2 11, change the APPLCOMPAT subsystem parameter value only after all applications use the SQL capabilities of Db2 11. For details, see "Enabling default application compatibility with Db2 11" on page 822.

Supported application compatibility levels

Db2 11 supports the following application compatibility levels in most contexts:

V11R1
: Compatibility with the behavior of Db2 11 new-function mode.

V10R1
: Compatibility with the behavior of DB2 10 new-function mode.

Example: V10R1 application compatibility

When you migrate a DB2 10 new-function mode environment to Db2 11 new-function mode, the initial application compatibility value is set to V10R1. The following example shows the results of using a V11R1 function while in Db2 11 new-function mode with application compatibility set to V10R1. In this example the CREATE TYPE statement is successful but the CREATE PROCEDURE statement results in SQLCODE -4743.

```
CREATE TYPE PHONENUMBERS AS VARCHAR(12) ARRAY ??(1000000??)
```

```
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

```
CREATE PROCEDURE FIND_CUSTOMERS(
  IN NUMBERS_IN KRAMSC01.PHONENUMBERS,
  IN AREA_CODE CHAR(3),
  OUT NUMBERS_OUT KRAMSC01.PHONENUMBERS)
    BEGIN
      SET NUMBERS_OUT =
        (SELECT ARRAY_AGG(T.NUM)
        FROM UNNEST(NUMBERS_IN) AS T(NUM)
        WHERE SUBSTR(T.NUM, 1, 3) = AREA_CODE);
    END
```

```
DSNT408I SQLCODE = -4743, ERROR:  ATTEMPT TO USE A FUNCTION WHEN THE
   APPLICATION COMPATIBILITY SETTING IS SET FOR A PREVIOUS LEVEL
```

The APPLCOMPAT bind value for the CREATE PROCEDURE statement is then set to V11R1 and result of the statement is then successful.

```
CREATE PROCEDURE FIND_CUSTOMERS(
  IN NUMBERS_IN KRAMSC01.PHONENUMBERS,
  IN AREA_CODE CHAR(3),
  OUT NUMBERS_OUT KRAMSC01.PHONENUMBERS)
  APPLCOMPAT V11R1
    BEGIN
      SET NUMBERS_OUT =
        (SELECT ARRAY_AGG(T.NUM)
        FROM UNNEST(NUMBERS_IN) AS T(NUM)
        WHERE SUBSTR(T.NUM, 1, 3) = AREA_CODE);
    END
```

```
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

Related reference
APPLCOMPAT bind option (Db2 Commands)
CURRENT APPLICATION COMPATIBILITY (Db2 SQL)
APPL COMPAT LEVEL field (APPLCOMPAT subsystem parameter) (Db2 Installation and Migration)
SYSPACKAGE catalog table (Db2 SQL)
SET CURRENT APPLICATION COMPATIBILITY (Db2 SQL)
Related information
Db2 11 for z/OS Technical Overview (IBM Redbooks)

V10R1 application compatibility level

When you set the application compatibility level to V10R1, applications that attempt to use functions and features that are introduced in Db2 11 or later might behave differently or receive an error.

When your Db2 11 environment is migrated to new-function mode, you can run individual applications with some of the features and behavior of DB2 10. Your applications can continue to experience V10R1 behavior while in Db2 11 new-function mode. Then, you can migrate each application to a new application compatibility value separately until all are migrated. If application compatibility is set to V10R1 and you attempt to use the new functions of a later version, SQL might behave differently or result in a negative SQLCODE, such as SQLCODE -4743.

PSPI

You can run package level accounting or monitor traces with IFCID 0239 and review field QPACINCOMPAT, which indicates an SQL incompatible change. If a trace is started for IFCID 0366 or IFCID 0376 and application compatibility is set for a previous version, details about features and functions that have a change in behavior are written in field QW0366FN or QW0376FN.

PSPI

A migrated Db2 11 environment in conversion mode behaves with V10R1 application compatibility. Application and SQL incompatibilities are described in the migration information for each version.

The following table shows many of the features and functions that are controlled by application compatibility, and the results if you specify V10R1. If a behavior difference is traced, then the IFCID trace function code is shown.

Table 134. Behavior of V10R1 application compatibility

Feature or Function	Result with V10R1 application compatibility	IFCID 0366 or IFCID 0376 trace function code
An SQL statement in a client application includes an unsupported conversion (from a string type to a numeric type or from a numeric type to a string type), and implicit casting is disabled (DDF_COMPATIBILITY is set to SP_PARMS_NJV or to DISABLE_IMPCAST_NJV).	SQLCODE -301	7"1" on page 821
A client application executes an SQL CALL statement to execute a Db2 for z/OS stored procedure. The DDF_COMPATIBILITY subsystem parameter is set to SP_PARMS_NJV for client applications other than Java applications, or SP_PARMS_JV for Java applications.	The data types of the data that is returned from the SQL CALL statement match the data types of the CALL statement arguments. This behavior is compatible with the behavior before Version 10.	8"1" on page 821
A client application accesses Db2 11 from an IBM Data Server Driver for JDBC and SQLJ client. The DDF_COMPATIBILITY subsystem parameter is set to IGNORE_TZ for Java applications.	The Db2 server ignores the TIMEZONE portion, appended by the IBM Data Server Driver for JDBC and SQLJ, of the value in the TIMESTAMP WITH TIMEZONE input to a TIMESTAMP target. This behavior is compatible with the behavior before DB2 10.	9
BIF_COMPATIBILITY is set to V9_TRIM, and input *string-expression* is EBCDIC mixed data for the RTRIM, LTRIM, or STRIP built-in function.	The DB2 9 version of SYSIBM.LTRIM(*string-expression*), SYSIBM.RTRIM(*string-expression*), or SYSIBM.STRIP(*string-expression*) is executed.	10
An implicit insert or update of an XML document node	SQLCODE -20345	1101
A predicate expression with an explicit cast or an operation with an invalid value that does not affect the results of XPath processing	SQLCODE -20345	1102
How the resource limit facility uses ASUTIME value for nested routines	SQLCODE -905 is issued only when the ASUTIME limit of the top-level calling package is encountered.	1103

Feature or Function	Result with V10R1 application compatibility	IFCID 0366 or IFCID 0376 trace function code
The lengths of values that are returned from CURRENT CLIENT_USERID, CURRENT CLIENT_WRKSTNNAME, CURRENT CLIENT_APPLNAME, or CURRENT CLIENT_ACCTNG special register are longer than the DB2 10 limits.	The special register values are truncated to the DB2 10 maximum lengths and padded with blanks	1104, 1105, 1106, 1107
A CAST(string as TIMESTAMP) specification with an input string of length of 8 or an input string of length 13	An explicit cast specification from string as TIMESTAMP interprets an 8-byte character string as a Store Clock value and a 13-byte string as a GENERATE_UNIQUE value. CAST result might be incorrect.	1109
Invocation of the SPACE or VARCHAR built-in function when the result is defined as VARCHAR(32765), VARCHAR(32766), or VARCHAR(32767)	No error	1110, 1111
Subsystem parameter XML_RESTRICT_EMPTY_TAG is set to YES, and an empty XML element is serialized as *<emptyElement></emptyElement>*	No error	1112
Specification of bind option DBPROTOCOL(DRDACBF)	DSNT298I	
A period specification that follows the name of a view in the FROM clause of a query	SQLCODE -4743	
A period clause that follows the name of a target view in an UPDATE or DELETE statement	SQLCODE -4743	
A SET CURRENT TEMPORAL SYSTEM_TIME statement	SQLCODE -4743	
A SET CURRENT TEMPORAL BUSINESS_TIME statement	SQLCODE -4743	
A reference to a global variable	SQLCODE -4743	
Use of array operations and built-in functions such as • Use of the UNNEST collection-derived-table • Use of the ARRAY_FIRST, ARRAY_LAST, ARRAY_NEXT, ARRAY_PRIOR, ARRAY_AGG, TRIM_ARRAY, CARDINALITY, MAX_CARDINALITY built-in functions • A SET assignment-statement of an array element as a target table • A CAST specification with a parameter marker as the source and an array as the data type	SQLCODE -4743	

Table 134. Behavior of V10R1 application compatibility (continued)

Feature or Function	Result with V10R1 application compatibility	IFCID 0366 or IFCID 0376 trace function code
An aggregate function that contains the keyword DISTINCT and references a column that is defined with a column mask	SQLCODE -20478	
An SQL statement contains the GROUP BY clause and references a column that is defined with a column mask	SQLCODE -20478	
An SQL statement contains the set operator UNION ALL or UNION DISTINCT and references a column that is defined with a column mask	SQLCODE -20478	
A reference to an alias for a sequence object	SQLCODE -4743	
A reference to an unqualified sequence that is not resolved to a public alias	SQLCODE -204	
A SELECT with a table function reference that includes a typed correlation clause	SQLCODE -4743	
A CALL statement that specifies an autonomous procedure	SQLCODE -4743	
The following datetime assignments: • A valid string representation of a timestamp to a date column • A valid string representation of a timestamp to a time column • A valid string representation of a date to a timestamp column	SQLCODE -180	

Notes:

1. PSPI To find details about the incompatible parameters, examine the contents of fields QW0366SC_Var, QW0366PR_Var, and QW0366INC_Var, or QW0376SC_Var, QW0376PR_Var, and QW0376INC_Var. See the *prefix*.SDSNIVPD(DSNWMSGS) file for more information. PSPI

Related concepts

Application and SQL release incompatibilities (Db2 for z/OS What's New?)

Managing application incompatibilities

Before you move an application to a new application compatibility level, you need to find application incompatibilities, adjust your applications for those incompatibilities, and verify that the incompatibilities no longer exist.

About this task

The following procedure explains how to find application incompatibilities that generate trace records. Other application incompatibilities generate SQL errors or Db2 error messages, but do not generate trace records. See "V10R1 application compatibility level" on page 818 for a complete list of application incompatibilities.

Procedure

1. Start a trace that includes IFCID 0239 to capture package information.

   For example, issue the following START TRACE command: GUPI

   ```
   -START TRACE(ACCTG) CLASS(7,8,10)
   ```

   GUPI

2. Examine the trace output.

   PSPI IFCID 0239 field QPACFLGS contains a bit that is on if a package contains incompatibilities. If this bit is off, no incompatibilities were detected, and you can skip the rest of the steps. If this bit is on, proceed to step 3. PSPI

3. Start a trace for IFCID 0376 to report incompatibility information about the packages.

   For example, issue the following START TRACE command: GUPI

   ```
   -START TRACE(PERFM) CLASS(32) IFCID(376)
   ```

   GUPI

4. Run the application.
5. Examine the trace output.

   PSPI IFCID 0376 fields contain information about the incompatibilities. Db2 writes a single trace record for each SQL statement that is incompatible with the subsequent Db2 release. See file *prefix*.SDSNIVPD(DSNWMSGS) for listings of the IFCID 0239 and 0376 trace records. PSPI

6. Revise the application to avoid any application incompatibilities.
7. Prepare the application for execution. When you bind the packages for the application, use the old APPLCOMPAT value.
8. Run the application.
9. Examine the trace output again to verify that the incompatibilities no longer exist.

What to do next

When the application runs at the old level with no application incompatibilities, rebind the package with the APPLCOMPAT value for the new release.

Related concepts

Performance trace (Db2 Performance)

V10R1 application compatibility level

When you set the application compatibility level to V10R1, applications that attempt to use functions and features that are introduced in Db2 11 or later might behave differently or receive an error.

Related reference

-START TRACE (Db2) (Db2 Commands)

Trace field descriptions (Db2 Performance)

Enabling default application compatibility with Db2 11

The APPLCOMPAT subsystem parameter specifies the default value of the APPLCOMPAT bind option. Before Db2 11 new-function mode, the APPLCOMPAT subsystem parameter must be set to V10R1. This setting ensures that existing SQL applications are bound for compatibility with the DB2 10 by default.

Before you begin

- Migrate your Db2 subsystem to new-function mode, as described in Migrating your Db2 subsystem to new-function mode (Db2 Installation and Migration).

- Take the following precautions to ensure that applications are ready with the Db2 11 application compatibility by default.
  - Identify and resolve all application incompatibilities of the higher level, as described in "Managing application incompatibilities" on page 821.
  - Rebind any packages that must continue to run at the lower application compatibility level and explicitly specify the APPLCOMPAT bind option for that level.

Procedure

To enable default application compatibility with Db2 11 new-function mode:

1. Change the APPLCOMPAT subsystem parameter setting. Set the value to V11R1.

   You can complete this step as described in Updating subsystem parameter and application default values (Db2 Installation and Migration), or by modifying your customized copy of the DSNTIJUZ job.
2. Run the first two job steps of DSNTIJUZ to rebuild your subsystem parameter (DSNZP*xxx*) module.
3. Use the -SET SYSPARM command or restart Db2.

Results

Future bind and rebind operations set the application compatibility level of the package to the APPLCOMPAT subsystem parameter value, if the APPLCOMPAT bind option is not specified. Packages that are bound or rebound at the higher level can begin use of SQL capabilities introduced at that level.

Related concepts

Application compatibility levels in Db2

The *application compatibility level* of your applications controls the adoption and use of new capabilities and enhancements, and the impact of incompatible changes. The advantage is that you can complete the Db2 11 migration process without the need to update your applications immediately.

Related reference

APPL COMPAT LEVEL field (APPLCOMPAT subsystem parameter) (Db2 Installation and Migration)

APPLCOMPAT bind option (Db2 Commands)

Chapter 9. Preparing an application to run on Db2 for z/OS

To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

Before you begin

To avoid rework, follow these steps:

1. Test your SQL statements by using SPUFI.
2. Compile your program with no SQL statements, and resolve all compiler errors.
3. Proceed with the preparation and the Db2 precompiler or with the host compiler that supports that Db2 coprocessor.

The following types of applications require different methods of program preparation:

- Applications that contain ODBC calls
- Applications in interpreted languages, such as REXX. For information about running REXX programs, which you do not prepare for execution, see "Running a Db2 REXX application" on page 924.
- Java applications, which can contain JDBC calls or embedded SQL statements

Procedure

You can perform these steps by using one of the following methods:

The following topics provide details on preparing and running a Db2 application:

a) "Processing SQL statements" on page 829
b) "Compiling and link-editing an application" on page 851
c) "Binding application packages and plans" on page 852
d) Chapter 10, "Running an application on Db2 for z/OS," on page 921

Binding a package is not necessary in all cases. These instructions assume that you bind some of your DBRMs into packages and include a package list in your plan.

If you use CICS, you might need additional steps; see:

- "Translating command-level statements in a CICS program" on page 838
- "Example of calling applications in a command procedure" on page 933

For more information about when to bind a package, see "Db2 program preparation overview" on page 881.

- **Preparing applications by using JCL procedures**

  A number of methods are available for preparing an application to run. You can:

  – Use Db2 interactive (DB2I) panels, which lead you step by step through the preparation process.
  – Submit a background job using JCL (which the program preparation panels can create for you).
  – Start the DSNH CLIST in TSO foreground or background.
  – Use TSO prompters and the DSN command processor.
  – Use JCL procedures added to your SYS1.PROCLIB (or equivalent) at Db2 installation time.
  – For C and C++ only, you can invoke the coprocessor from UNIX System Services. If the DBRM is generated in a HFS file, you can also use the command line processor to bind the resulting DBRM.

Optionally, you can also copy the DBRM into a partitioned data set member by using the oput and oget commands and then bind it by using conventional JCL.

This topic describes how to use JCL procedures to prepare a program. For information about using the DB2I panels, see Chapter 9, "Preparing an application to run on Db2 for z/OS," on page 825.

- **Preparing applications by the Db2 Program Preparation panels**

  If you develop programs using TSO and ISPF, you can prepare them to run by using the Db2 Program Preparation panels. These panels guide you step by step through the process of preparing your application to run. Other ways of preparing a program to run are available, but using Db2 Interactive (DB2I) is the easiest because it leads you automatically from task to task.

  Important: If your C++ program satisfies both of the following conditions, you must use a JCL procedure to prepare it:

  – The program consists of more than one data set or member.

  – More than one data set or member contains SQL statements.

  To prepare an application by using the Db2 Program Preparation panels:

  a. If you want to display or suppress message IDs during program preparation, specify one of the following commands on the ISPF command line:

     TSO PROFILE MSGID
        Message IDs are displayed

     TSO PROFILE NOMSGID
        Message IDs are supressed

  b. Open the DB2I Primary Option Menu.

  c. Select the option that corresponds to the Program Preparation panel.

  d. Complete the Program Preparation panel and any subsequent panels. After you complete each panel, DB2I automatically displays the next appropriate panel.

- **Preparation guidelines for DL/I batch programs**

  Use the following guidelines when you prepare a program to access Db2 and DL/I in a batch program:

  – "Processing SQL statements by using the Db2 precompiler" on page 830

  – "Binding a batch program" on page 865

  – "Compiling and link-editing an application" on page 851

  – "Loading and running a batch program" on page 927

Related concepts
The Db2 command line processor (Db2 Commands)
TSO attachment facility (Introduction to Db2 for z/OS)
Related reference
The DB2I primary option menu (Introduction to Db2 for z/OS)
DSNH (TSO CLIST) (Db2 Commands)

Setting the DB2I defaults

When you use the Db2 Interactive (DB2I) panels to prepare an application, you can specify the default values that DB2I is to use. These defaults values can include the default application language and default JCL JOB statement. Otherwise, DB2I uses the system default values that were set at installation time.

Procedure

As DB2I leads you through a series a panels, enter the default values that you want on the following panels when they are displayed.

Table 135. DB2I panels to use to set default values

If you want to set the following default values...	Use this panel
• subsystem ID • number of additional times to attempt to connect to Db2 • programming language • number of lines on each page of listing or SPUFI output • lowest level of message to return to you during the BIND phase • SQL string delimiter for COBOL programs • how to represent decimal separators • smallest value of the return code (from precompile, compile, link-edit, or bind) that prevents later steps from running • default number of input entry rows to generate on the initial display of ISPF panels • user ID to associate with the trusted connection for the current DB2I session	DB2I Defaults Panel 1 panel
• default JOB statement • symbol used to delimit a string in a COBOL statement in a COBOL application • whether DCLGEN generates a picture clause that has the form PIC G(n) DISPLAY-1 or PIC N(n).	DB2I Defaults Panel 2 panel

Table 135. DB2I panels to use to set default values (continued)	
If you want to set the following default values...	**Use this panel**
The following package and plan characteristics • isolation level • whether to check authorization at run time or at bind time • when to release locks on resources • whether to obtain EXPLAIN information about how SQL statements in the plan or package execute • whether you need data currency for ambiguous cursors opened at remote locations • whether to use parallel processing • whether Db2 determines access paths at bind time and again at execution time • whether to defer preparation of dynamic SQL statements • whether Db2 keeps dynamic SQL statements after commit points • the application encoding scheme • whether you want to use optimization hints to determine access paths • when Db2 writes the changes for updated group buffer pool-dependent pages • whether run time (RUN) or bind time (BIND) rules apply to dynamic SQL statements at run time • whether to continue to create a package after finding SQL errors (packages only) • when to acquire locks on resources (plans only) • whether a CONNECT (Type 2) statement executes according to Db2 rules (Db2) or the SQL standard (STD). (plans only) • which remote connections end during a commit or a rollback (plans only)	Defaults for Bind Package panel Defaults for Bind Plan panel

Related reference

DB2I Defaults Panel 1
DB2I Defaults Panel 1 lets you change many of the system default values that were set at Db2 installation time.

DB2I Defaults Panel 2
After you press Enter on the DB2I Defaults Panel 1, the DB2I Defaults Panel 2 is displayed. If you chose IBMCOB as the language on the DB2I Defaults Panel 1, three fields are displayed. Otherwise, only the first field is displayed.

Defaults for Bind Package and Defaults for Rebind Package panels
These DB2I panels lets you change your defaults for BIND PACKAGE and REBIND PACKAGE options.

Defaults for Bind Plan and Defaults for Rebind Plan panels

These DB2I panels let you change your defaults for BIND PLAN and REBIND PLAN options.

Processing SQL statements

The first step in preparing an SQL application to run is to process the SQL statements in the program. To process the statements, use either the Db2 precompiler or the Db2 coprocessor. During this step, the SQL statements are replaced with calls to Db2 language interface modules, and a DBRM is created.

About this task

For assembler or Fortran applications, use the Db2 precompiler to prepare the SQL statements.

For C, C++, COBOL, or PL/I applications, you can use one of the following techniques to process SQL statements:

- Use the Db2 precompiler before you compile your program.

  You can use this technique with any supported version of C or C++, COBOL, or PL/I.
- Invoke the Db2 coprocessor for the host language that you are using as you compile your program. You can use the Db2 coprocessor with C, C++, COBOL, and PL/I host compilers. To invoke the Db2 coprocessor, specify the SQL compiler option followed by its suboptions, which are those options that are defined for the Db2 precompiler. Some Db2 precompiler options are ignored.
  - For C or C++, you need IBM z/OS Version 1 Release 8 C/C++ or later. For C and C++, you can also invoke the coprocessor from UNIX System Services on z/OS to generate a DBRM in either a partitioned data set or an HFS file.
  - For COBOL, you need Enterprise COBOL for z/OS Version 3 Release 4 or later to use this technique.
  - For PL/I, you need Enterprise PL/I for z/OS Version 3 Release 4 or later to use this technique.

CICS: If the application contains CICS commands, you must translate the program before you compile it.

Db2 version in DSNHDECP module

When you process SQL statements in your program, if the Db2 version in DSNHDECP is the default system-provided version, Db2 issues a warning and processing continues. In this case, ensure that the information in DSNHDECP that Db2 uses accurately reflects your environment.

SQL statement processing

Because most compilers do not recognize SQL statements, you can prevent compiler errors by using either the Db2 precompiler or the Db2 coprocessor.

The precompiler scans the program and returns modified source code, which you can then compile and link edit. The precompiler also produces a DBRM (database request module). You can bind this DBRM to a package using the BIND subcommand. When you complete these steps, you can run your Db2 application.

Alternatively, you can use the Db2 coprocessor for the host language. The Db2 coprocessor performs Db2 precompiler functions at compile time. When you use the Db2 coprocessor, the compiler (rather than the precompiler) scans the program and returns the modified source code. The Db2 coprocessor also produces a DBRM.

Related concepts

Using the Db2 C/C++ precompiler (XL C/C++ Programming Guide)

Db2 coprocessor (Enterprise COBOL for z/OS Programming Guide)

Differences between the Db2 precompiler and the Db2 coprocessor
The Db2 precompiler and Db2 coprocessor have architectural differences. You cannot switch from one to the other without considering those differences and adjusting your program accordingly.

Db2 program preparation overview

Before you can run an application program on Db2 for z/OS, you need to prepare it. To prepare the program, create a load module, possibly one or more packages, and an application plan.

Program directories for Db2 11 (Db2 for z/OS in IBM Documentation)

Related tasks

Translating command-level statements in a CICS program
You can translate CICS applications with the CICS command language translator as a part of the program preparation process. CICS command language translators are available only for assembler, C, COBOL, and PL/I languages.

Related reference

Enterprise COBOL for z/OS

Processing SQL statements by using the Db2 precompiler

The Db2 precompiler scans a program and copies all of the SQL statements and host variable information into a DBRM (database request module). The precompiler also returns source code that has been modified so that the SQL statements do not cause errors when you compile the program.

About this task

After the SQL statements and host variable information are copied into a DBRM and the modified source code is returned, you can compile and link-edit this modified source code.

Before you run the Db2 precompiler, use DCLGEN to obtain accurate SQL DECLARE TABLE statements. The precompiler checks table and column references against SQL DECLARE TABLE statements in the program, not the actual tables and columns.

Db2 does not need to be active when you precompile your program.

You do not need to precompile the program on the same Db2 subsystem on which you bind the DBRM and run the program. You can bind a DBRM and run it on a Db2 subsystem at the previous release level, if the original program does not use any properties of Db2 that are unique to the current release. You can also run applications on the current release that were previously bound on subsystems at the previous release level.

Procedure

To process SQL statements by using the Db2 precompiler:

1. Ensure that your program is ready to be processed by the Db2 precompiler by performing the following actions.

   For information about the criteria for programs that are passed to the precompiler, see "Input to the Db2 precompiler" on page 833.
2. If you plan to run multiple precompilation jobs and are not using the DFSMSdfp partitioned data set extended (PDSE), change the Db2 language preparation procedures (DSNHCOB, DSNHCOB2, DSNHICOB, DSNHFOR, DSNHC, DSNHPLI, DSNHASM, DSNHSQL) to specify the DISP=OLD parameter instead of the DISP=SHR parameter.

   The Db2 language preparation procedures in job DSNTIJMV use the DISP=OLD parameter to enforce data integrity. However, the installation process converts the DISP=OLD parameter for the DBRM library data set to DISP=SHR, which can cause data integrity problems when you run multiple precompilation jobs.
3. Start the precompile process by using one of the following methods:

   - DB2I panels. Use the **Precompile** panel or the Db2 **Program Preparation** panels. For details, see "DB2I panels that are used for program preparation" on page 888.
   - The DSNH command procedure (a TSO CLIST). For details, see DSNH (TSO CLIST) (Db2 Commands).
   - JCL procedures that are supplied with Db2. For details, see "Db2-supplied JCL procedures for preparing an application" on page 885.

Recommendation: Specify the SOURCE and XREF precompiler options to get complete diagnostic output from the Db2 precompiler. This output is useful if you need to precompile and compile program source statements several times before they are error-free and ready to link-edit.

The output that is returned from the Db2 precompiler is described in "Output from the Db2 precompiler" on page 835.

What to do next

Preparing a program with object-oriented extensions by using JCL

If your C++ or Enterprise COBOL for z/OS program satisfies both of these conditions, you need special JCL to prepare it:

- The program consists of more than one data set or member.
- More than one data set or member contains SQL statements.

You must precompile the contents of each data set or member separately, but the prelinker must receive all of the compiler output together.

JCL procedure DSNHCPP2, which is in member DSNTIJMV of data set DSN1110.SDSNSAMP, shows you one way to do this for C++.

Precompiling a batch program

When you add SQL statements to an application program, you must precompile the application program and bind the resulting DBRM into a package, as described in Chapter 9, "Preparing an application to run on Db2 for z/OS," on page 825.

Related concepts

DCLGEN (declarations generator)
Your program should declare the tables and views that it accesses. The Db2 declarations generator, DCLGEN, produces these DECLARE statements for C, COBOL, and PL/I programs, so that you do not need to code the statements yourself. DCLGEN also generates corresponding host variable structures.

Related reference

DSNH (TSO CLIST) (Db2 Commands)

Data sets that the precompiler uses

When you invoke the precompiler you need to provide data sets that contain input for the precompiler, such as the host programming statements and SQL statements. You also need to provide data sets where the precompiler can store its output, such as the modified source code and diagnostics messages.

Table 136. DD statements and data sets that the Db2 precompiler uses

DD statement	Data set description	Required?
DBRMLIB	Output data set, which contains the SQL statements and host variable information that the Db2 precompiler extracted from the source program. It is called Database Request Module (DBRM). This data set becomes the input to the Db2 bind process. The DCB attributes of the data set are RECFM FB, LRECL 80. DBRMLIB has to be a PDS and a member name must be specified. You can use IEBCOPY, IEHPROGM, TSO commands, COPY and DELETE, or PDS management tools for maintaining the data set.	Yes

Table 136. DD statements and data sets that the Db2 precompiler uses (continued)

DD statement	Data set description	Required?
STEPLIB	Step library for the job step. In this DD statement, you can specify the name of the library for the precompiler load module, DSNHPC, and the name of the library for your Db2 application programming defaults member, DSNHDECP. **Recommendation:** Always use the STEPLIB DD statement to specify the library where your Db2 DSNHDECP module resides to ensure that the proper application defaults are used by the Db2 precompiler. The library that contains your Db2 DSNHDECP module needs to be allocated ahead of the prefix.SDSNLOAD library.	No, but recommended
SYSCIN	Output data set, which contains the modified source that the Db2 precompiler writes out. This data set becomes the input data set to the compiler or assembler. This data set must have attributes RECFM F or FB, and LRECL 80. SYSCIN can be a PDS or a sequential data set. If a PDS is used, the member name must be specified.	Yes
SYSIN	Input data set, which contains statements in the host programming language and embedded SQL statements. This data set must have the attributes RECFM F or FB, LRECL 80. SYSIN can be a PDS or a sequential data set. If a PDS is used, the member name must be specified.	Yes
SYSLIB	INCLUDE library, which contains additional SQL and host language statements. The Db2 precompiler includes the member or members that are referenced by SQL INCLUDE statements in the SYSIN input from this DD statement. Multiple data sets can be specified, but they must be partitioned data sets with attributes RECFM F or FB, LRECL 80. SQL INCLUDE statements cannot be nested.	No
SYSPRINT	Output data set, which contains the output listing from the Db2 precompiler. This data set must have an LRECL of 133 and a RECFM of FBA. SYSPRINT must be a sequential data set	Yes
SYSTERM	Terminal output file, which contains diagnostic messages from the Db2 precompiler. The DCB attributes of the data set are determined by the z/OS system. SYSTERM must be a sequential data set.	No

DD statement	Data set description	Required?
SYSUT1 and SYSUT2	Internal work files that the precompiler uses to store temporary information as it converts embedded SQL statements to host language statements. Precompilation of assembler and PL/I source code uses only the SYSUT1 data set. The default SPACE parameter values in the Db2-supplied program preparation procedures (DSNHASM, DSNHC, DSNHCPP, DSNHCPP2, DSNHICOB, DSNHPLI, and DSNHFOR) are adequate in most cases. If your application program contains a large number of embedded SQL statements, you might need to increase those values.	No, unless you need to override the default SPACE parameter values.

Table 136. DD statements and data sets that the Db2 precompiler uses (continued)

Related reference

SPACE Parameter (MVS JCL Reference)

Input to the Db2 precompiler

The primary input for the precompiler consists of statements in the host programming language and embedded SQL statements.

You can use the SQL INCLUDE statement to get secondary input from the include library, SYSLIB. The SQL INCLUDE statement reads input from the specified member of SYSLIB until it reaches the end of the member.

Another preprocessor, such as the PL/I macro preprocessor, can generate source statements for the precompiler. Any preprocessor that runs before the precompiler must be able to pass on SQL statements. Similarly, other preprocessors can process the source code, after you precompile and before you compile or assemble.

Input to the Db2 precompiler has the following restrictions:

- The size of a source program that Db2 can precompile is limited by the region size and the virtual memory available to the precompiler. These amounts vary with each system installation.
- The forms of source statements that can pass through the precompiler are limited. For example, constants, comments, and other source syntax that are not accepted by the host compilers (such as a missing right brace in C) can interfere with precompiler source scanning and cause errors. To check for such unacceptable source statements, run the host compiler before the precompiler. You can ignore the compiler error messages for SQL statements or comment out the SQL statements. After the source statements are free of unacceptable compiler errors, you can then uncomment any SQL statements that you previously commented out and continue with the normal Db2 program preparation process for that host language.
- You must write host language statements and SQL statements using the same margins, as specified in the precompiler option MARGINS.
- The input data set, SYSIN, must have the attributes RECFM F or FB, LRECL 80.
- SYSLIB must be a partitioned data set, with attributes RECFM F or FB, LRECL 80.
- Input from the INCLUDE library cannot contain other precompiler INCLUDE statements.

Starting the precompiler dynamically when using JCL procedures

You can call the precompiler from an assembler program by using a macro.

About this task

You can call the precompiler from an assembler program by using one of the macro instructions ATTACH, CALL, LINK, or XCTL.

To call the precompiler, specify DSNHPC as the entry point name. You can pass three address options to the precompiler; the following topics describe their formats. The options are addresses of:

- A precompiler option list
- A list of alternative DD names for the data sets that the precompiler uses
- A page number to use for the first page of the compiler listing on SYSPRINT

Related reference

Using X-macros (MVS Assembler Services Reference)

Precompiler option list format

When you call the precompiler, you can specify a number of options, in a list, for SQL statement processing. You must specify that option list in a particular format.

The option list must begin on a 2-byte boundary. The first 2 bytes contain a binary count of the number of bytes in the list (excluding the count field). The remainder of the list is EBCDIC and can contain precompiler option keywords, separated by one or more blanks, a comma, or both.

DD name list format

When you call the precompiler, you can specify a list of alternative DD names for the data sets that the precompiler uses. You must specify this list in a particular format.

The DD name list must begin on a 2-byte boundary. The first 2 bytes contain a binary count of the number of bytes in the list (excluding the count field). Each entry in the list is an 8-byte field, left-justified, and padded with blanks if needed.

The following table gives the following sequence of entries:

Table 137. DDNAME list entries

Entry	Standard ddname	Usage
1	Not applicable	
2	Not applicable	
3	Not applicable	
4	SYSLIB	Library input
5	SYSIN	Source input
6	SYSPRINT	Diagnostic listing
7	Not applicable	
8	SYSUT1	Work data
9	SYSUT2	Work data
10	Not applicable	
11	Not applicable	
12	SYSTERM	Diagnostic listing
13	Not applicable	

Table 137. DDNAME list entries (continued)

Entry	Standard ddname	Usage
14	SYSCIN	Changed source output
15	Not applicable	
16	DBRMLIB	DBRM output

Page number format

When you call the precompiler, you can specify a page number to use for the first page of the compiler listing on SYSPRINT. You must specify this page number in a particular format.

A 6-byte field beginning on a 2-byte boundary contains the page number. The first 2 bytes must contain the binary value 4 (the length of the remainder of the field). The last 4 bytes contain the page number in character or zoned-decimal format.

The precompiler adds 1 to the last page number that is used in the precompiler listing and puts this value into the page-number field before returning control to the calling routine. Thus, if you call the precompiler again, page numbering is continuous.

Output from the Db2 precompiler

The major output from the Db2 precompiler is a database request module (DBRM). However, the Db2 precompiler also produces modified source statements, a list of source statements, a list of statements that refer to host names and columns, and diagnostics.

Specifically, the precompiler produces the following types of output:

listing output
: The Db2 precompiler writes the following information in the SYSPRINT data set:

- Precompiler source listing

  If the Db2 precompiler option SOURCE is specified, a source listing is produced. The source listing includes precompiler source statements, with line numbers that are assigned by the precompiler.
- Precompiler diagnostics

  The precompiler produces diagnostic messages that include precompiler line numbers of statements that have errors.
- Precompiler cross-reference listing

  If the Db2 precompiler option XREF is specified, a cross-reference listing is produced. The cross-reference listing shows the precompiler line numbers of SQL statements that refer to host names and columns.

The SYSPRINT data set has an LRECL of 133 and a RECFM of FBA. This data set uses the CCSID of the source program. Statement numbers in the output of the precompiler listing are displayed as they appear in the listing.

Terminal diagnostics
: If a terminal output file, SYSTERM, exists, the Db2 precompiler writes diagnostic messages to it. A portion of the source statement accompanies the messages in this file. You can often use the SYSTERM file instead of the SYSPRINT file to find errors. This data set uses EBCDIC.

Modified source statements
: The Db2 precompiler writes the source statements that it processes to SYSCIN, the input data set to the compiler or assembler. This data set must have attributes RECFM F or FB, and LRECL 80. The modified source code contains calls to the Db2 language interface. The SQL statements that the calls replace appear as comments. This data set uses the CCSID of the source program.

Database request modules
: The database request module (DBRM) is a data set that contains the SQL statements and host variable information that is extracted from the source program, along with information that identifies

the program and ties the DBRM to the translated source statements. It becomes the input to the bind process.

The data set requires space to hold all the SQL statements plus space for each host variable name and some header information. The header information alone requires approximately two records for each DBRM, 20 bytes for each SQL record, and 6 bytes for each host variable.

For an exact format of the DBRM, see the DBRM mapping macros, DSNXDBRM and DSNXNBRM, in library *prefix*.SDSNMACS. The DCB attributes of the data set are RECFM FB, LRECL 80. The precompiler sets the characteristics. You can use IEBCOPY, IEHPROGM, TSOCOPY and DELETE commands, or other PDS management tools for maintaining these data sets.

Restriction: Do not modify the contents of the DBRM. If you do, unpredictable results can occur. Db2 does not support modified DBRMs.

In a DBRM, the SQL statements and the list of host variable names use the UTF-8 character encoding scheme.

All other character fields in a DBRM use EBCDIC. The current release marker (DBRMMRIC) in the header of a DBRM is marked according to the release of the precompiler, regardless of the value of NEWFUN.

Processing SQL statements by using the Db2 coprocessor

As an alternative to the Db2 precompiler, you can use theDb2 coprocessor to process SQL statements. The Db2 coprocessor performs Db2 precompiler functions at compile time.

About this task

Exception: For PL/I, the Db2 coprocessor is called from the PL/I SQL preprocessor instead of the compiler.

The Db2 coprocessor has fewer restrictions on SQL programs than the Db2 precompiler. When you process SQL statements with the Db2 coprocessor, you can do the following things in your program:

- Use fully qualified names for structured host variables.
- Include SQL statements at any level of a nested program, instead of in only the top-level source file. (Although you can include SQL statements at any level of a nested program, you must compile the entire program as one unit.)
- Use nested SQL INCLUDE statements.
- **For C or C++ programs only:** Write applications with variable length format.
- **For C or C++ programs only:** Use codepage-dependent characters, such as left and right brackets, without using tri-graph notation when the programs use different code pages.

Procedure

To process SQL statements by using the Db2 coprocessor, take one of the following actions:

- Submit a JCL job to process that SQL statement. Include the following information:
  - Specify the SQL compiler option when you compile your program:

    The SQL compiler option indicates that you want the compiler to invoke the Db2 coprocessor. Specify a list of SQL processing options in parentheses after the SQL keyword. Table 139 on page 842 lists the options that you can specify.

    For COBOL and PL/I, enclose the list of SQL processing options in single or double quotation marks. For PL/I, separate options in the list by a comma, blank, or both, as shown in the following examples:

    C/C++

    ```
    SQL(APOSTSQL STDSQL(NO))
    ```

COBOL

```
SQL("APOSTSQL STDSQL(NO)")
```

PL/I

```
PP(SQL("APOSTSQL,STDSQL(NO)")
```

– For PL/I programs that use BIGINT or LOB data types, specify the following compiler options when you compile your program:

```
LIMITS(FIXEDBIN(63), FIXEDDEC(31))
```

– If needed, increase the user's region size so that it can accommodate more memory for the Db2 coprocessor.

– Include DD statements for the following data sets in the JCL for your compile step:

- Db2 load library (*prefix*.SDSNLOAD)

  The Db2 coprocessor calls Db2modules to process the SQL statements. You therefore need to include the name of the Db2 load library data set in the STEPLIB concatenation for the compiler step.

- DBRM library

  The Db2 coprocessor produces a DBRM. DBRMs and the DBRM library are described in "Output from the Db2 precompiler" on page 835. You need to include a DBRMLIB DD statement that specifies the DBRM library data set.

- Library for SQL INCLUDE statements

  If your program contains SQL INCLUDE *member-name* statements that specify secondary input to the source program, you need to also specify the data set for *member-name*. Include the name of the data set that contains *member-name* in the SYSLIB concatenation for the compiler step.

- **For C or C++ programs only:** Invoke the Db2 coprocessor from UNIX System Services on z/OS.

  If you invoke the Db2 coprocessor from UNIX System Services, you can choose to have the DBRM generated in a partitioned data set or an HFS file.

  When you invoke the Db2 coprocessor, specify the SQL compiler option. The SQL compiler option indicates that you want the compiler to invoke the Db2 coprocessor. Specify a list of SQL processing options in parentheses after the SQL keyword. For the list of options that you can specify, see SQL processing options.

  The file name for the DBRM is determined as described in DRBMLIB. For C and C++, you can specify one of the following items:

  – The name of a partitioned data set. The following example invokes the C/C++ Db2 coprocessor to compile (with the c89 compiler) a sample C program and requests that the resulting DBRM is stored in the test member of the `userid.dbrmlib.data` data set:

```
c89 -Wc,"sql,dbrmlib(//'userid.dbrmlib.data(test)'),langlvl(extended)" -c t.c
```

  – The name of an HFS file. The name can be qualified, partially qualified, or unqualified. The file path can contain a maximum of 1024 characters, and the file name can contain a maximum of 255 characters. The first 8 characters of the file name, not including the file extension, must be unique within the file system.

  For example, assume that your directory structure is `/u/USR001/c/example` and that your current working directory is `/u/USR001/c`. The following table shows examples of how to specify the HFS file names with the DBRMLIB option and how the file names are resolved.

Table 138. How to specify HFS files to store DBRMs

If you specify...	The DBRM is generated in...
`dbrmlib(/u/USR001/sample.dbrm)`	`/u/USR001/sample.dbrm`
`dbrmlib(example/sample.dbrm)`	`/u/USR001/c/example/sample.dbrm`
`dbrmlib(../sample.dbrm)`	`/u/USR001/sample.dbrm`
`dbrmlib(sample.dbrm)`	`/u/USR001/c/sample.dbrm`

The following example invokes the Db2 coprocessor to compile (with the c89 compiler) a sample C program and requests that the resulting DBRM is stored in the file `test.dbrm` in the `tmp` directory:

```
c89 -Wc,"sql,dbrmlib(/tmp/test.dbrm),langlvl(extended)" -c t.c
```

If you request that the DBRM be generated in an HFS file, you can bind the resulting DBRM by using the command line processor BIND command. For more information about using the command line processor BIND command, see “Binding a DBRM that is in an HFS file to a package or collection” on page 855. Optionally, you can also copy the DBRM into a partitioned data set member by using the oput and oget commands and then bind the DBRM by using conventional JCL.

Support for compiling a COBOL program that includes SQL from an assembler program

The COBOL compiler provides a facility that enables you to invoke the COBOL compiler by using an assembler program.

If you intend to use the Db2 coprocessor and start the COBOL compiler from an assembler program as part of your Db2 application preparation, you can use the SQL compiler option and provide the alternate DBRMLIB DD name the same way that you can specify other alternate DD names. The Db2 coprocessor creates the DBRM member according to your DBRM PDS library and the DBRM member that you specified using the alternate DBRMLIB DD name.

To use the alternate DBRMLIB DD name, Enterprise COBOL V4.1 and above is required.

Related reference

Starting the compiler from an assembler program

Translating command-level statements in a CICS program

You can translate CICS applications with the CICS command language translator as a part of the program preparation process. CICS command language translators are available only for assembler, C, COBOL, and PL/I languages.

Procedure

Prepare your CICS program in either of these sequences:

Sequence	Remarks
a. **Db2 precompiler** b. **CICS Command Language Translator.**	This sequence is the preferred method of program preparation and the one that the DB2I Program Preparation panels support. If you use the DB2I panels for program preparation, you can specify translator options automatically, rather than needing to provide a separate option string.

Sequence	Remarks
a. **CICS command language translator** b. **Db2 precompiler**	This sequence results in a warning message from the CICS translator for each EXEC SQL statement that it encounters. The warning messages have no effect on the result. If you are using double-byte character sets (DBCS), precompiling is recommended before translating, as described previously.

Use the Db2 precompiler before the CICS translator to prevent the precompiler from mistaking CICS translator output for graphic data.

If your source program is in COBOL, you must specify a string delimiter that is the same for the Db2 precompiler, COBOL compiler, and CICS translator. The defaults for the Db2 precompiler and COBOL compiler are not compatible with the default for the CICS translator.

If the SQL statements in your source program refer to host variables that a pointer stored in the CICS TWA addresses, you must make the host variables addressable to the TWA before you execute those statements. For example, a COBOL application can issue the following statement to establish addressability to the TWA:

```
EXEC CICS ADDRESS
     TWA (address-of-twa-area)
END-EXEC
```

You can run CICS applications only from CICS address spaces. This restriction applies to the RUN option on the second program DSN command processor. All of those possibilities occur in TSO.

To prepare an application program, you can append JCL from a job that is created by the Db2 Program Preparation panels to the JCL for the CICS command language translator. To run the prepared program under CICS, you might need to define programs and transactions to CICS. Your system programmer must make the appropriate CICS resource or table entries.

prefix.SDSNSAMP contains examples of the JCL that is used to prepare and run a CICS program that includes SQL statements. The set of JCL includes:

- PL/I macro phase
- Db2 precompiling
- CICS Command Language Translation
- Compiling of the host language source statements
- Link-editing of the compiler output
- Binding of the DBRM
- Running of the prepared application.

Related reference

Sample applications in CICS
A set of Db2 sample applications run in the CICS environment.

Related information

Resource definition (CICS Transaction Server for z/OS)

Differences between the Db2 precompiler and the Db2 coprocessor

The Db2 precompiler and Db2 coprocessor have architectural differences. You cannot switch from one to the other without considering those differences and adjusting your program accordingly.

Recommendation: Use the coprocessor instead of the precompiler when using Unicode variables in COBOL or PL/I applications.

Depending on whether you use the Db2 precompiler or the Db2 coprocessor, ensure that you account for the following differences:

- Differences in handling source CCSIDs:

  The Db2 precompiler and Db2 coprocessor convert the SQL statements of your source program to UTF-8 for parsing.

  The precompiler or Db2 coprocessor uses the source CCSID(*n*) value to convert from that CCSID to CCSID 1208 (UTF-8). The CCSID value must be an EBCDIC CCSID. If you want to prepare a source program that is written in a CCSID that cannot be directly converted to or from CCSID 1208, you must create an indirect conversion.

- Differences in handling host variable CCSIDs:

  – **COBOL:**

    Db2 precompiler:
    The Db2 precompiler sets CCSIDs for alphanumeric host variables only when the program includes an explicit DECLARE :hv VARIABLE statement.

    Db2 coprocessor:
    The COBOL compiler with National Character Support always sets CCSIDs for alphanumeric variables, including host variables that are used within SQL, to the source CCSID. Alternatively, you can specify that you want the COBOL Db2 coprocessor to handle CCSIDs the same way as the precompiler.

    Recommendation: If you have problems with host variable CCSIDs, use the Db2 precompiler or change your application to include the DECLARE :hv VARIABLE statement to overwrite the CCSID that is specified by the COBOL compiler.

    Example: Assume that Db2 has mapped a FOR BIT DATA column to a host variable in the following way:

```
01  hv1  pic x(5).
01  hv2  pic x(5).

EXEC SQL  CREATE TABLE T1 (colwbit  char(5) for bit data,
                           rowid    char(5))   END-EXEC.

   EXEC SQL
   INSERT INTO T1 VALUES (:hv1, :hv2)
   END-EXEC.
```

    Db2 precompiler: In the modified source from the Db2 precompiler, hv1 and hv2 are represented to Db2 through SQLDA in the following way, without CCSIDs:

```
for hv1: NO CCSID

20 SQL-PVAR-NAMEL1  PIC S9(4) COMP-4 VALUE +0.
20 SQL-PVAR-NAMEC1  PIC X(30) VALUE ' '.

for hv2:  NO CCSID

20 SQL-PVAR-NAMEL2  PIC S9(4) COMP-4 VALUE +0.
20 SQL-PVAR-NAMEC2  PIC X(30) VALUE ' '
```

    Db2 coprocessor: In the modified source from the Db2 coprocessor with the National Character Support for COBOL, hv1 and hv2 are represented to Db2 in the following way, with CCSIDs: (Assume that the source CCSID is 1140.)

```
for hv1 and hv2, the value for CCSID is set to '1140' ('474'x) in input SQLDA
of the INSERT statement.

'7F00000474000000007F'x
```

    To ensure that no discrepancy exists between the column with FOR BIT DATA and the host variable with CCSID 1140, add the following statement for :hv1 or use the Db2 precompiler:

```
EXEC SQL DECLARE : hv1 VARIABLE FOR BIT DATA END-EXEC.

 for hv1 declared with for bit data. The value in  SQL---AVAR-NAME-DATA  is
  set to  'FFFF'x for CCSID instead of '474x'.
```

```
'7F0000FFFF000000007F'x  <<= with DECLARE :hv1 VARIABLE FOR BIT DATA
vs.
'7F00000474000000007F'x   <<= without
```

– **PL/I**

Db2 coprocessor:
You can specify whether CCSIDs are to be associated with host variables by using the following PL/I SQL preprocessor options:

CCSID0
Specifies that the PL/I SQL preprocessor is not to set the CCSIDs for all host variables unless they are defined with the SQL DECLARE :hv VARIABLE statement.

NOCCSID0
Specifies that the PL/I SQL preprocessor is to set the CCSIDs for all host variables.

Related concepts
z/OS Unicode Services User's Guide and Reference
Related reference
Descriptions of SQL processing options
You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

Enterprise COBOL for z/OS
SQL preprocessor options (PL/I) (Enterprise PL/I for z/OS Programming Guide:)

Options for SQL statement processing

Use SQL processing options to specify how the Db2 precompiler and the Db2 coprocessor interpret and process input, and how they present output.

If you are using the Db2 precompiler, specify SQL processing options in one of the following ways:

- With DSNH operands
- With the PARM option of the EXEC JCL statement
- On DB2I panels

If you are using the Db2 coprocessor, specify SQL processing options in one of the following ways:

- For C or C++, specify the options as the argument of the SQL compiler option.
- For COBOL, specify the options as the argument of the SQL compiler option.
- For PL/I, specify the options as the argument of the PP(SQL('option,...')) compiler option.

For examples of how to specify the Db2 coprocessor options, see "Processing SQL statements by using the Db2 coprocessor" on page 836

Db2 assigns default values for any SQL processing options for which you do not explicitly specify a value. Those defaults are the values that are specified on the APPLICATION PROGRAMMING DEFAULTS installation panels.

Descriptions of SQL processing options

You can specify any SQL processing options regardless of whether you use the Db2 precompiler or the Db2 coprocessor. However, the Db2 coprocessor might ignore certain options because host language compiler options exist that provide the same information.

The following table shows the options that you can specify when you use the Db2 precompiler or Db2 coprocessor. The table also includes abbreviations for those options and indicates which options are ignored for a particular host language or by the Db2 coprocessor. This table uses a vertical bar (|) to separate mutually exclusive options, and brackets ([ ]) to indicate that you can sometimes omit the enclosed option.

Table 139. SQL processing options

Option keyword	Meaning
APOST[1]	Indicates that the Db2 precompiler is to use the apostrophe (') as the string delimiter in host language statements that it generates. This option is not available in all languages. APOST and QUOTE are mutually exclusive options. The default is in the field STRING DELIMITER on Application Programming Defaults Panel 1 during installation. If STRING DELIMITER is the apostrophe ('), APOST is the default.
APOSTSQL	Recognizes the apostrophe (') as the string delimiter and the double quotation mark (") as the SQL escape character within SQL statements. APOSTSQL and QUOTESQL are mutually exclusive options. The default is in the field SQL STRING DELIMITER on Application Programming Defaults Panel 1 during installation. If SQL STRING DELIMITER is the apostrophe ('), APOSTSQL is the default.
ATTACH(TSO|CAF| RRSAF)	Specifies the attachment facility that the application uses to access Db2 TSO, CAF, or RRSAF that load the attachment facility can use this option to specify the correct attachment facility, instead of coding a dummy DSNHLI entry point. This option is not available for Fortran applications. The default is ATTACH(TSO).
CCSID(*n*)	Specifies the numeric value *n* of the CCSID in which the source program is written. The number *n* must be an EBCDIC CCSID. The default setting is the EBCDIC system CCSID as specified on the panel DSNTIPF during installation. The Db2 coprocessor uses the following process to determine the CCSID of the source statements: 1. If the CCSID of the source program is specified by a compiler option, such as the COBOL CODEPAGE compiler option, the Db2 coprocessor uses that CCSID. If you also specify the CCSID suboption of the SQL compiler option that is different from the CCSID compiler option, a warning is returned, and the CCSID suboption value is not used. 2. If the CCSID is not specified by a compiler option: a. If the CCSID suboption of the SQL compiler option is specified and contains a valid EBCDIC CCSID, that CCSID is used. b. If the CCSID suboption of the SQL compiler option is not specified, and the compiler supports an option for specifying the CCSID, such as the COBOL CODEPAGE compiler option, the default for the CCSID compiler option is used. c. If the CCSID suboption of the SQL compiler option is not specified, and the compiler does not support an option for specifying the CCSID, the default CCSID from DSNHDECP or a user-specified application defaults module is used. d. If the CCSID suboption of the SQL option is specified and contains an invalid CCSID, compilation terminates. CCSID supersedes the GRAPHIC and NOGRAPHIC SQL processing options. If you specify CCSID(1026) or CCSID(1155), the Db2 coprocessor does not support the code point 'FC'X for the double quotation mark (").

Table 139. SQL processing options (continued)	
Option keyword	**Meaning**
COMMA	Recognizes the comma (,) as the decimal point indicator in decimal or floating point literals in the following cases: • For static SQL statements in COBOL programs • For dynamic SQL statements, when the value of installation parameter DYNRULS is NO and the package or plan that contains the SQL statements has DYNAMICRULES bind, define, or invoke behavior. COMMA and PERIOD are mutually exclusive options. The default (COMMA or PERIOD) is chosen under DECIMAL POINT IS on Application Programming Defaults Panel 1 during installation.
CONNECT(2|1) CT(2|1)	Determines whether to apply type 1 or type 2 CONNECT statement rules. CONNECT(2) Default: Apply rules for the CONNECT (Type 2) statement CONNECT(1) Apply rules for the CONNECT (Type 1) statement If you do not specify the CONNECT option when you precompile a program, the rules of the CONNECT (Type 2) statement apply.
DATE(ISO|USA |EUR|JIS|LOCAL)	Specifies that date output should always be returned in a particular format, regardless of the format that is specified as the location default. The default is specified in the field DATE FORMAT on Application Programming Defaults Panel 2 during installation. The default format is determined by the installation defaults of the system where the program is bound, not by the installation defaults of the system where the program is precompiled. You cannot use the LOCAL option unless you have a date exit routine.
DEC(15|31) DEC15 | DEC31 D15.*s* | D31.*s*	Specifies the maximum precision for decimal arithmetic operations. The default is in the field DECIMAL ARITHMETIC on Application Programming Defaults Panel 1 during installation. If the form D*pp.s* is specified, *pp* must be either 15 or 31, and *s*, which represents the minimum scale to be used for division, must be a number between 1 and 9.
DECP(*name*)	*name* represents the 1 to 8 character name of the application defaults data-only load module that is to be used. The default name DSNHDECP is used if this parameter is omitted.
FLAG(I|W|E|S)[1]	Suppresses diagnostic messages below the specified severity level (Informational, Warning, Error, and Severe error for severity codes 0, 4, 8, and 12 respectively). The default setting is FLAG(I).
FLOAT(S390|IEEE)	Determines whether the contents of floating-point host variables in assembler, C, C++, or PL/I programs are in IEEE floating-point format or z/Architecture hexadecimal floating-point format. Db2 ignores this option if the value of HOST is anything other than ASM, C, CPP, or PLI. The default setting is FLOAT(S390).

<table>
<caption>Table 139. SQL processing options (continued)</caption>
<tr><th>Option keyword</th><th>Meaning</th></tr>
<tr><td>GRAPHIC</td><td>This option is no longer used for SQL statement processing. Use the CCSID option instead.

Indicates that the source code might use mixed data, and that X'0E'and X'0F' are special control characters (shift-out and shift-in) for EBCDIC data.

GRAPHIC and NOGRAPHIC are mutually exclusive options. The default (GRAPHIC or NOGRAPHIC) is specified in the field MIXED DATA on Application Programming Defaults Panel 1 during installation.</td></tr>
<tr><td>HOST[1](ASM|C[(FOLD)]|
CPP[(FOLD)]|
IBMCOB|
PLI|
FORTRAN|
SQL|
SQLPL)</td><td>Defines the host language that contains the SQL statements.

Use IBMCOB for Enterprise COBOL for z/OS.

For C, specify:
• C if you do not want Db2 to fold lowercase letters in SBCS SQL ordinary identifiers to uppercase
• C(FOLD) if you want Db2 to fold lowercase letters in SBCS SQL ordinary identifiers to uppercase

For C++, specify:
• CPP if you do not want Db2 to fold lowercase letters in SBCS SQL ordinary identifiers to uppercase
• CPP(FOLD) if you want Db2 to fold lowercase letters in SBCS SQL ordinary identifiers to uppercase

For SQL procedural language, specify:
• SQL, to perform syntax checking and conversion to a generated C program for an external SQL procedure.
• SQLPL, to perform syntax checking for a native SQL procedure.

If you omit the HOST option, the Db2 precompiler issues a level-4 diagnostic message and uses the default value for this option.

The default is in the field LANGUAGE DEFAULT on Application Programming Defaults Panel 1 during installation.

This option also sets the language-dependent defaults.</td></tr>
<tr><td>LEVEL[(aaaa)]
L</td><td>Defines the level of a module, where aaaa is any alphanumeric value of up to seven characters. This option is not recommended for general use, and the DSNH CLIST and the DB2I panels do not support it.

For assembler, C, C++, Fortran, and PL/I, you can omit the suboption (aaaa). The resulting consistency token is blank. For COBOL, you need to specify the suboption.</td></tr>
<tr><td>LINECOUNT[1](n)
LC</td><td>Defines the number of lines per page to be n for the Db2 precompiler listing. This includes header lines that are inserted by the Db2 precompiler. The default setting is LINECOUNT(60).</td></tr>
</table>

Table 139. SQL processing options (continued)	
Option keyword	**Meaning**
MARGINS[1](m,n[,c]) MAR	Specifies what part of each source record contains host language or SQL statements. For assembler, this option also specifies where column continuations begin. The first option (m) is the beginning column for statements. The second option (n) is the ending column for statements. The third option (c) specifies where assembler continuations begin. Otherwise, the Db2 precompiler places a continuation indicator in the column immediately following the ending column. Margin values can range 1 - 80. Default values depend on the HOST option that you specify. The DSNH CLIST and the DB2I panels do not support this option. In assembler, the margin option must agree with the ICTL instruction, if presented in the source.
NEWFUN(V*n*)	The NEWFUN processing option is deprecated. Indicates whether to accept the function syntax that is new for Db2 11. **NEWFUN(V11)** Specifies that any syntax up to Db2 11 is allowed. **NEWFUN(V10)** Specifies that any syntax up to DB2 10 is allowed. **NEWFUN(V9)** Specifies that any syntax up to DB2 9 is allowed. DB2 9 is supported, but causes the precompilation process to support only a DB2 9 level of function. **NEWFUN(V8)** Specifies that any syntax up to DB2 Version 8 is allowed. V8 is supported, but causes the precompilation process to support only a V8 level of function. The NEWFUN option applies only to the precompilation process by either the precompiler or the Db2 coprocessor, regardless of the current migration mode. You are responsible for ensuring that you bind the resulting DBRM on a subsystem in the correct migration mode.
NOFOR	In static SQL, eliminates the need for the FOR UPDATE or FOR UPDATE OF clause in DECLARE CURSOR statements. When you use NOFOR, your program can make positioned updates to any columns that the program has Db2 authority to update. When you do not use NOFOR, if you want to make positioned updates to any columns that the program has Db2 authority to update, you need to specify FOR UPDATE with no column list in your DECLARE CURSOR statements. The FOR UPDATE clause with no column list applies to static or dynamic SQL statements. Regardless of whether you use NOFOR, you can specify FOR UPDATE OF with a column list to restrict updates to only the columns that are named in the clause, and you can specify the acquisition of update locks. You imply NOFOR when you use the option STDSQL(YES). If the resulting DBRM is very large, you might need extra storage when you specify NOFOR or use the FOR UPDATE clause with no column list.

Table 139. SQL processing options (continued)

Option keyword	Meaning
NOGRAPHIC	This option is no longer used for SQL statement processing. Use the CCSID option instead. Indicates the use of X'0E'and X'0F' in a string, but not as control characters. GRAPHIC and NOGRAPHIC are mutually exclusive options. The default (GRAPHIC or NOGRAPHIC) is specified in the field MIXED DATA on Application Programming Defaults Panel 1 during installation. The NOGRAPHIC option applies to only EBCDIC data.
NOOPTIONS NOOPTN	Suppresses the Db2 precompiler options listing.
NOPADNTSTR	Indicates that output host variables that are NUL-terminated strings are **not** padded with blanks. That is, additional blanks are not inserted before the NUL-terminator is placed at the end of the string. PADNTSTR and NOPADNTSTR are mutually exclusive options. The default (PADNTSTR or NOPADNTSTR) is specified in the field PAD NUL-TERMINATED on Application Programming Defaults Panel 2 during installation. This option applies to only C and C++ applications.
NOSOURCE[2] NOS	Suppresses the Db2 precompiler source listing. This is the default.
NOXREF	Suppresses the Db2 precompiler cross-reference listing. This is the default.
ONEPASS ON	Processes in one pass, to avoid the additional processing time for making two passes. Declarations must appear before SQL references. Default values depend on the HOST option specified. ONEPASS and TWOPASS are mutually exclusive options.
OPTIONS[1] OPTN	Lists Db2 precompiler options. This is the default.
PADNTSTR	Indicates that output host variables that are NUL-terminated strings are padded with blanks with the NUL-terminator placed at the end of the string. PADNTSTR and NOPADNTSTR are mutually exclusive options. The default (PADNTSTR or NOPADNTSTR) is specified in the field PAD NUL-TERMINATED on Application Programming Defaults Panel 2 during installation. This option applies to only C and C++ applications.

Table 139. SQL processing options (continued)

<table>
<tr><th>Option keyword</th><th>Meaning</th></tr>
<tr><td>PERIOD</td><td>Recognizes the period (.) as the decimal point indicator in decimal or floating point literals in the following cases:
• For static SQL statements in COBOL programs
• For dynamic SQL statements, when the value of installation parameter DYNRULS is NO and the package or plan that contains the SQL statements has DYNAMICRULES bind, define, or invoke behavior.
COMMA and PERIOD are mutually exclusive options. The default (COMMA or PERIOD) is specified in the field DECIMAL POINT IS on Application Programming Defaults Panel 1 during installation.</td></tr>
<tr><td>QUOTE[1]
Q</td><td>Indicates that the Db2 precompiler is to use the quotation mark (") as the string delimiter in host language statements that it generates.
QUOTE is valid only for COBOL applications. QUOTE is not valid for either of the following combinations of precompiler options:
• CCSID(1026) and HOST(IBMCOB)
• CCSID(1155) and HOST(IBMCOB)
The default is specified in the field STRING DELIMITER on Application Programming Defaults Panel 1 during installation. If STRING DELIMITER is the double quotation mark (") or DEFAULT, QUOTE is the default.
APOST and QUOTE are mutually exclusive options.</td></tr>
<tr><td>QUOTESQL</td><td>Recognizes the double quotation mark (") as the string delimiter and the apostrophe (') as the SQL escape character within SQL statements. This option applies only to COBOL.
The default is specified in the field SQL STRING DELIMITER on Application Programming Defaults Panel 1 during installation. If SQL STRING DELIMITER is the double quotation mark (") or DEFAULT, QUOTESQL is the default.
APOSTSQL and QUOTESQL are mutually exclusive options.</td></tr>
<tr><td>SOURCE[1]
S</td><td>Lists Db2 precompiler source and diagnostics.</td></tr>
<tr><td>SQL(ALL|DB2)</td><td>Indicates whether the source contains SQL statements other than those recognized by Db2 for z/OS.
SQL(ALL) is recommended for application programs whose SQL statements must execute on a server other that Db2 for z/OS using DRDA access. SQL(ALL) indicates that the SQL statements in the program are not necessarily for Db2 for z/OS. Accordingly, the SQL statement processor then accepts statements that do not conform to the Db2 syntax rules. The SQL statement processor interprets and processes SQL statements according to distributed relational database architecture (DRDA) rules. The SQL statement processor also issues an informational message if the program attempts to use IBM SQL reserved words as ordinary identifiers. SQL(ALL) does not affect the limits of the SQL statement processor.
SQL(Db2), the default, means to interpret SQL statements and check syntax for use by Db2 for z/OS. SQL(Db2) is recommended when the database server is Db2 for z/OS.</td></tr>
</table>

Table 139. SQL processing options (continued)

Option keyword	Meaning
STDSQL(NO|YES)[3]	Indicates to which rules the output statements should conform. STDSQL(YES)[3] indicates that the precompiled SQL statements in the source program conform to certain rules of the SQL standard. STDSQL(NO) indicates conformance to Db2 rules. The default is specified in the field STD SQL LANGUAGE on Application Programming Defaults Panel 2 during installation. STDSQL(YES) automatically implies the NOFOR option.
TIME(ISO|USA|EUR|JIS|LOCAL)	Specifies that time output always return in a particular format, regardless of the format that is specified as the location default. The default is specified in the field TIME FORMAT on Application Programming Defaults Panel 2 during installation. The default format is determined by the installation defaults of the system where the program is bound, not by the installation defaults of the system where the program is precompiled. You cannot use the LOCAL option unless you have a time exit routine.
TWOPASS TW	Processes in two passes, so that declarations need not precede references. Default values depend on the HOST option that is specified. ONEPASS and TWOPASS are mutually exclusive options. For the Db2 coprocessor, you can specify the TWOPASS option for only PL/I applications. For C/C++ and COBOL applications, the Db2 coprocessor uses the ONEPASS option.
VERSION(*aaaa*|AUTO)	Defines the version identifier of a package, program, and the resulting DBRM. A version identifier is an SQL identifier of up to 64 EBCDIC bytes. When you specify VERSION, the SQL statement processor creates a version identifier in the program and DBRM. This affects the size of the load module and DBRM. Db2 uses the version identifier when you bind the DBRM to a package. If you do not specify a version at precompile time, an empty string is the default version identifier. If you specify AUTO, the SQL statement processor uses the consistency token to generate the version identifier. If the consistency token is a timestamp, the timestamp is converted into ISO character format and is used as the version identifier. The timestamp that is used is based on the store clock value.
XREF[5]	Includes a sorted cross-reference listing of symbols that are used in SQL statements in the listing output.

Table 139. SQL processing options (continued)

Option keyword	Meaning

Notes:

1. The Db2 coprocessor ignores this option when the Db2 coprocessor is invoked by the compiler to prepare the application.
2. This option is always in effect when the Db2 coprocessor is invoked by the compiler to prepare the application.
3. You can use STDSQL(86) as in prior releases of Db2. The SQL statement processor treats it the same as STDSQL(YES).
4. Precompiler options do not affect ODBC behavior.
5. The Db2 coprocessor ignores this option when the Db2 coprocessor is invoked by the compiler to prepare the application. However, if you are using PL/I V4.1 or later, it is supported.

Related concepts

Precision for operations with decimal numbers
Db2 accepts two sets of rules for determining the precision and scale of the result of an operation with decimal numbers.

Datetime values (Db2 SQL)

Related tasks

Creating a package version
If you want to run different versions of a program without needing to make changes to the associated application plan, use package versions. This technique is useful if you need to make changes to your program without causing an interruption to the availability of the program.

Setting the program level
The program level defines the level for a particular module. This information is stored in the consistency token, which is in an internal Db2 format. Overriding the program level in the consistency token is possible, if needed, but generally not recommended.

Related reference

Defaults for SQL processing options
Some SQL statement processing options have default values that are based on values that are specified on the DB2I Application Programming Defaults panels.

Defaults for SQL processing options

Some SQL statement processing options have default values that are based on values that are specified on the DB2I Application Programming Defaults panels.

The following table shows those options and defaults.

Table 140. IBM-supplied installation default SQL statement processing options. The installer can change these defaults.

Install option	Install default	Equivalent SQL statement processing option	Available SQL statement processing options
STRING DELIMITER	quotation mark (")	QUOTE	APOSTQUOTE
SQL STRING DELIMITER	quotation mark (")	QUOTESQL	APOSTSQLQUOTESQL
DECIMAL POINT IS	PERIOD	PERIOD	COMMAPERIOD
DATE FORMAT	ISO	DATE(ISO)	DATE(ISO|USA| EUR|JIS| LOCAL)

Table 140. IBM-supplied installation default SQL statement processing options. The installer can change these defaults. *(continued)*			
Install option	**Install default**	**Equivalent SQL statement processing option**	**Available SQL statement processing options**
DECIMAL ARITHMETIC	DEC15	DEC(15)	DEC(15|31)
MIXED DATA	NO	CCSID(n)	CCSID(n)
LANGUAGE DEFAULT	COBOL	HOST(COBOL)	HOST(ASM|C[(FOLD)]|CPP[(FOLD)]|IBMCOB|FORTRAN|PLI)
STD SQL LANGUAGE	NO	STDSQL(NO)	STDSQL(YES|NO|86)
TIME FORMAT	ISO	TIME(ISO)	TIME(IS|USA|EUR| JIS|LOCAL)

Notes: For dynamic SQL statements, another application programming default, USE FOR DYNAMICRULES, determines whether Db2 uses the application programming default or the SQL statement processor option for the following installation options:

- STRING DELIMITER
- SQL STRING DELIMITER
- DECIMAL POINT IS
- DECIMAL ARITHMETIC

If the value of USE FOR DYNAMICRULES is YES, dynamic SQL statements use the application programming defaults. If the value of USE FOR DYNAMICRULES is NO, dynamic SQL statements in packages or plans with bind, define, and invoke behavior use the SQL statement processor options.

Some SQL statement processor options have default values based on the host language. Some options do not apply to some languages. The following table shows the language-dependent options and defaults.

Table 141. Language-dependent Db2 precompiler options and defaults

HOST value	Defaults
ASM	APOST[1], APOSTSQL[1], PERIOD[1], TWOPASS, MARGINS(1,71,16)
C or CPP	APOST[1], APOSTSQL[1], PERIOD[1], ONEPASS, MARGINS(1,72)
IBMCOB	QUOTE[2], QUOTESQL[2], PERIOD, ONEPASS[1], MARGINS(8,72)[1]
FORTRAN	APOST[1], APOSTSQL[1], PERIOD[1], ONEPASS[1], MARGINS(1,72)[1]
PLI	APOST[1], APOSTSQL[1], PERIOD[1], ONEPASS, MARGINS(2,72)
SQL or SQLPL	APOST[1], APOSTSQL[1], PERIOD[1], ONEPASS, MARGINS(1,72)

Notes:

1. Forced for this language; no alternative is allowed.
2. The default is chosen on Application Programming Defaults Panel 1 during installation. The IBM-supplied installation defaults for string delimiters are QUOTE (host language delimiter) and QUOTESQL (SQL escape character). The installer can replace the IBM-supplied defaults with other defaults. The precompiler options that you specify override any defaults that are in effect.

SQL statement processing defaults for dynamic statements

Generally, dynamic statements use the defaults that are specified during installation. However, if the value of application defaults module parameter DYNRULS is NO, you can use these options for dynamic SQL statements in packages or plans with bind, define, or invoke behavior:

- COMMA or PERIOD
- APOST or QUOTE
- APOSTSQL or QUOTESQL
- DEC(15) or DEC(31)

Related concepts

Dynamic rules options for dynamic SQL statements
The DYNAMICRULES bind option and the runtime environment determine the rules for the dynamic SQL attributes.

SQL options for DRDA access

Certain SQL statement processing options are relevant when you prepare a package to be run with DRDA access.

The following SQL statement processing options are relevant for DRDA access:

CONNECT
: Use **CONNECT(2)**, explicitly or by default.

  CONNECT(1) causes your CONNECT statements to allow only the restricted function known as "remote unit of work". Be particularly careful to avoid CONNECT(1) if your application updates more than one DBMS in a single unit of work.

SQL
: Use **SQL(ALL)** explicitly for a package that runs on a server that *is not* Db2 for z/OS. The precompiler then accepts any statement that obeys DRDA rules.

  Use **SQL(Db2)**, explicitly or by default, if the server is Db2 for z/OS only. The precompiler then rejects any statement that does not obey the rules of Db2 for z/OS.

Compiling and link-editing an application

If you use the Db2 precompiler, your next step in the program preparation process is to compile and link-edit your program. As with the precompile step, you have a choice of methods.

About this task

You can use one of the following methods to compile and link-edit an application:

- DB2I panels. For details, see “DB2I panels that are used for program preparation” on page 888.
- The DSNH command procedure (a TSO CLIST). For details, see DSNH (TSO CLIST) (Db2 Commands).
- JCL procedures supplied with Db2. For details, see “Db2-supplied JCL procedures for preparing an application” on page 885.
- JCL procedures supplied with a host language compiler.

If you use the Db2 coprocessor, you process SQL statements as you compile your program. For programs other than C and C++ programs, you must use JCL procedures when you use the Db2 coprocessor. For C and C++ programs, you can use either JCL procedures or UNIX System Services on z/OS to invoke the Db2 coprocessor.

The purpose of the link-edit step is to produce an executable load module. To enable your application to interface with the Db2 subsystem, you must use a link-edit procedure that builds a load module that satisfies environment-specific requirements.

TSO and batch
Include the Db2 TSO attachment facility language interface module (DSNELI) or Db2 call attachment facility language interface module (DSNALI) or the Universal Language Interface module (DSNULI).

IMS
Include the Db2 IMS language interface module (DFSLI000), which contains the DSNHLI entry point. Also, the IMS RESLIB must precede the SDSNLOAD library in the link list, JOBLIB, or STEPLIB concatenations.

IMS and Db2 share a common alias name, DSNHLI, for the language interface module. You must do the following when you concatenate your libraries:

- If you use IMS, be sure to concatenate the IMS library first so that the application program compiles with the correct IMS version of DSNHLI.
- If you run your application program only under Db2, be sure to concatenate the Db2 library first.

CICS
Include the Db2 CICS language interface module (DSNCLI) or the Universal Language Interface module (DSNULI). You can link DSNCLI with your program in either 24-bit or 31-bit addressing mode (AMODE=31), but DSNULI must be linked with your program in 31-bit addressing mode (AMODE=31). If your application runs in 31-bit addressing mode, you should link-edit the DSNCLI or DSNULI stub to your application with the attributes AMODE=31 and RMODE=ANY so that your application can run above the 16-MB line.

You also need the CICS EXEC interface module that is appropriate for the programming language. CICS requires that this module be the first control section (CSECT) in the final load module.

The size of the executable load module that is produced by the link-edit step varies depending on the values that the SQL statement processor inserts into the source code of the program.

Link-editing a batch program
Db2 has language interface routines for each unique supported environment. Db2 requires the IMS language interface routine for DL/I batch. You need to have DFSLI000 link-edited with the application program.

Related concepts

Universal language interface (DSNULI)
The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

Related tasks

Preparing an application to run on Db2 for z/OS
To prepare and run applications that contain embedded static SQL statements or dynamic SQL statements, you must precompile, compile, link-edit, and bind them.

Related reference

DSNH (TSO CLIST) (Db2 Commands)

Related information

CICS program preparation steps (CICS Transaction Server for z/OS)

Binding application packages and plans

You must bind the DBRM that is produced by the SQL statement processor to a package before your Db2 application can run. The bind process establishes a relationship between an application program and its relational data.

Before you begin

You must have appropriate privileges. For more information, see Privileges required for handling plans and packages (Managing Security)

About this task

During the precompilation process, the Db2 precompiler produces both modified source code and a database request module (DBRM) for each application program. The modified source code must be compiled and link-edited before the program can be run. DBRMs must be bound to a package. You can then associate that package with a particular application plan.

During the bind process, Db2 also completes the following actions:

- Validates object references in the SQL statements of the program, such as table, view, and column names, against the Db2 catalog. Because the bind process occurs before program execution, errors are detected and can be corrected before the program is executed.
- Verifies the authorization of the bind process to specify the program owner and the authorization of the specified owner to access data that is requested by SQL statements in the program.
- Selects the access paths that Db2 uses to access data for the program. Db2 considers factors such as table size, available indexes, and others, when selecting the access paths.

When determining the maximum size of a plan, you must consider several physical limitations, including the time required to bind the plan, the size of the EDM pool, and fragmentation. As a general rule, the EDM pool should be at least 10 times the size of the largest DBD or plan, whichever is greater.

Each package that you bind can contain only one DBRM.

Exception: You do not need to bind a DBRM if the only SQL statement in the program is SET CURRENT PACKAGESET.

Because you do not need a plan or package to execute the SET CURRENT PACKAGESET statement, the ENCODING bind option does not affect the SET CURRENT PACKAGESET statement. An application that needs to provide a host variable value in an encoding scheme other than the system default encoding scheme must use the DECLARE VARIABLE statement to specify the encoding scheme of the host variable.

You must bind plans locally, regardless of whether they reference packages that run remotely. However, you must bind the packages that run at remote locations at those remote locations.

From a Db2 requester, you can run a plan by specifying it in the RUN subcommand, but you cannot run a package directly. You must include the package in a plan and then run the plan.

Tip: Develop a naming convention and strategy for the most effective and efficient use of your plans and packages.

Procedure

To bind application programs, take the following actions.

1. To bind individual DBRMs into packages, use BIND PACKAGE commands with ACTION(REPLACE). Packages provide the flexibility for you to test different versions of a program without having to rebind everything in the application plan.

   C or C++ programs only: For programs whose corresponding DBRMs are in HFS files, you can use the command line processor to bind the DBRMs to packages. Optionally, you can also copy the DBRM into a partitioned data set member by using the **`oput`** and **`oget`** commands and then bind it by using conventional JCL.

   To create new trigger packages for existing triggers, you must re-create the trigger that is associated with the package. For more information, see "Trigger packages" on page 161.
2. To designate packages in application plans, use the BIND PLAN command with ACTION(REPLACE). Plans can specify packages, collections of packages, or a combination of these elements. If you specify one or more DBRMs to include in the plan (by using the MEMBER option of BIND PLAN), Db2 automatically binds those DBRMs into packages and then binds those packages into the plan. The plan contains information about the designated packages and about the data that the application programs intend to use. The plan is stored in the Db2 catalog.

Related concepts

Package copies for plan management (Db2 Performance)

Automatic rebinds
Automatic rebinds (sometimes called "autobinds") occur when an authorized user runs a package or plan and the runtime structures in the plan or package cannot be used. This situation usually results from changes to the attributes of the data on which the package or plan depends, or changes to the environment in which the package or plan runs.

Related tasks

Binding a DBRM that is in an HFS file to a package or collection
If DBRMs are in z/OS UNIX HFS files, you can use the command line processor to bind the DBRMs to packages at the target Db2 server. Optionally, you can also copy the DBRM into a partitioned data set member by using the TSO/E oput and oget commands and then bind the DBRM by using conventional JCL.

Related reference

BIND PACKAGE (DSN) (Db2 Commands)

BIND PLAN (DSN) (Db2 Commands)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Creating a package version

If you want to run different versions of a program without needing to make changes to the associated application plan, use package versions. This technique is useful if you need to make changes to your program without causing an interruption to the availability of the program.

About this task

You can create a different package version for each version of the program. Each package has the same package name and collection name, but a different version number is associated with each package. The plan that includes that package includes all versions of that package. Thus, you can run a program that is associated with any one of the package versions without having to rebind the application plan, rename the plan, or change any RUN subcommands that use it.

Procedure

To create a package version:

1. Precompile your program with the option VERSION(*version-identifier*).
2. Bind the resulting DBRM with the same collection name and package name as any existing versions of that package. When you run the program, Db2 uses the package version that you specified when you precompiled it.

Example

Suppose that you bound a plan with the following statement:

```
BIND PLAN (PLAN1) PKLIST (COLLECT.*)
```

The following steps show how to create two versions of a package, one for each of two programs.

Step number	For package version 1	For package version 2
1	Precompile program 1. Specify VERSION(1).	Precompile program version 2. Specify VERSION(2).
2	Bind the DBRM with the collection name COLLECT and the package name PACKA.	Bind the DBRM with the collection name COLLECT and package name PACKA.
3	Link-edit program 1 into your application.	Link-edit program 2 into your application.
4	Run the application; it uses program 1 and PACKA, VERSION 1.	Run the application; it uses program 2 and PACKA, VERSION 2.

Binding a DBRM that is in an HFS file to a package or collection

If DBRMs are in z/OS UNIX HFS files, you can use the command line processor to bind the DBRMs to packages at the target Db2 server. Optionally, you can also copy the DBRM into a partitioned data set member by using the TSO/E oput and oget commands and then bind the DBRM by using conventional JCL.

About this task

Only DBRMs for C and C++ programs can be generated to HFS files.

Restrictions:

You cannot specify the REBIND command with the command line processor. Alternatively, specify the BIND command with the ACTION(REPLACE) option.

You cannot specify the FREE PACKAGE command with the command line processor. Alternatively, specify the DROP PACKAGE statement to drop the existing packages.

Procedure

To bind a DBRM that is in an HFS file to a package or collection:

1. Invoke the command line processor and connect to the target Db2 server.
2. Specify the BIND command with the appropriate options.

Related concepts

The Db2 command line processor (Db2 Commands)

Related tasks

Processing SQL statements by using the Db2 coprocessor
As an alternative to the Db2 precompiler, you can use theDb2 coprocessor to process SQL statements. The Db2 coprocessor performs Db2 precompiler functions at compile time.

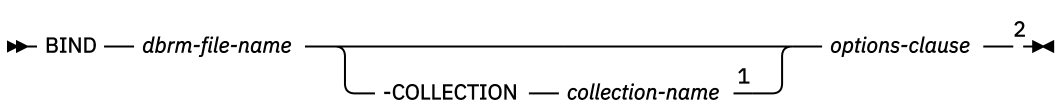
Related reference

Command line processor BIND command
Use the command line processor BIND command to bind DBRMs that are in z/OS UNIX HFS files to packages.

Command line processor BIND command

Use the command line processor BIND command to bind DBRMs that are in z/OS UNIX HFS files to packages.

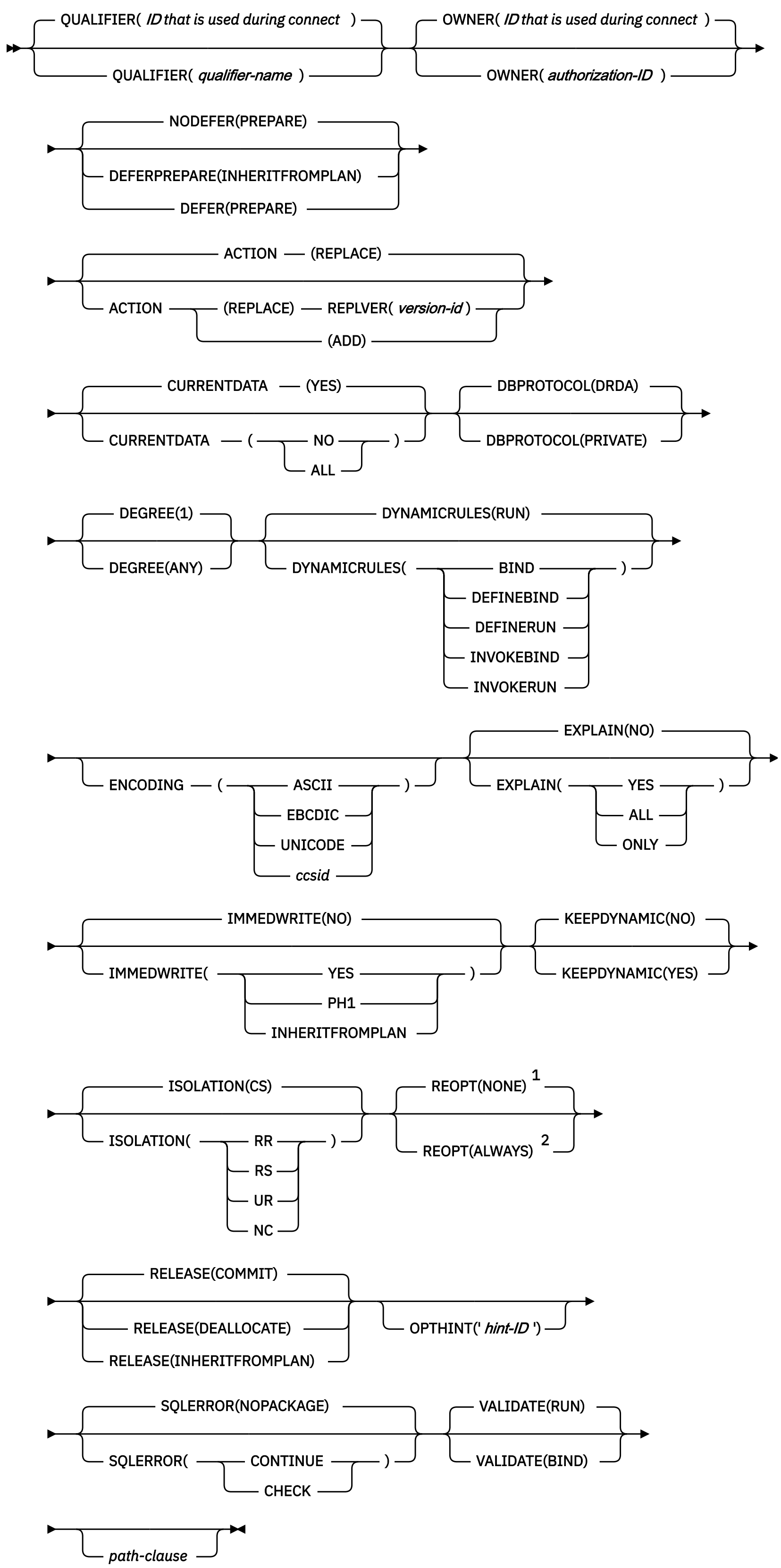
The following diagram shows the syntax for the command line processor BIND command.



Notes:

[1] If you do not specify a collection, Db2 uses NULLID.
[2] You can specify the options after *collection-name* in any order.

options-clause:

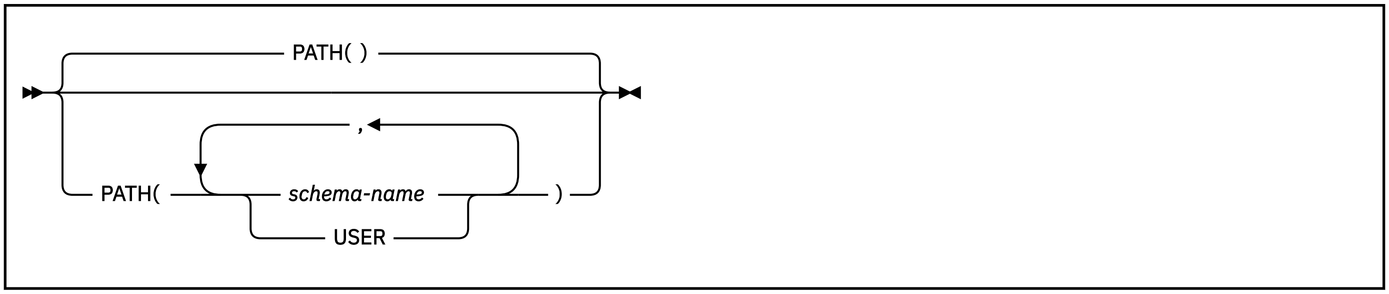


Notes:

[1] You can specify NOREOPT(VARS) as a synonym of REOPT(NONE).

[2] You can specify REOPT(VARS) as a synonym of REOPT(ALWAYS).

path-clause:



The following options are unique to this diagram:

CURRENTDATA (ALL)
Specifies that for all cursors data currency is required and block fetching is inhibited.

SQLERROR(CHECK)
Specifies that the command line processor is to only check for SQL errors in the DBRM. No package is generated.

IMMEDWRITE(PH1)
Specifies that normal write activity is done. This option is equivalent to IMMEDWRITE(NO).

EXPLAIN(ALL)
Specifies that Db2 is to insert information into the appropriate EXPLAIN tables. This option is equivalent to EXPLAIN (YES).

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Binding an application plan

An application plan can include package lists.

Procedure

To bind an application plan, use the BIND PLAN subcommand with at least one of the following options:

MEMBER
Specify this option to bind DBRMs to a package and then bind the package list to a plan. After the keyword MEMBER, specify the member names of the DBRMS.

PKLIST
Specify this option to include package lists in the plan. After the keyword PKLIST, specify the names of the packages to include in the package list. To include an entire collection of packages in the list, use an asterisk after the collection name. For example, `PKLIST(GROUP1.*)`.

Specifying the package list for the PKLIST option of BIND PLAN
The order in which you specify packages in a package list can affect run time performance. Searching for the specific package involves searching the Db2 directory, which can be costly. When you use *collection-id*.* with the PKLIST keyword, you should specify first the collections in which Db2 is most likely to find a package.

For example, assume that you perform the following bind:

```
BIND PLAN (PLAN1) PKLIST (COLL1.*, COLL2.*, COLL3.*, COLL4.*)
```

Then you execute program PROG1. Db2 does the following package search:

a. Checks to see if program PROG1 is bound as part of the plan
b. Searches for COLL1.PROG1.*timestamp*
c. If it does not find COLL1.PROG1.*timestamp*, searches for COLL2.PROG1.*timestamp*
d. If it does not find COLL2.PROG1.*timestamp*, searches for COLL3.PROG1.*timestamp*

e. If it does not find COLL3.PROG1.*timestamp*, searches for COLL4.PROG1.*timestamp*.

When both special registers CURRENT PACKAGE PATH and CURRENT PACKAGESET contain an empty string

If you do not set these special registers, Db2 searches for a DBRM or a package in one of these sequences:

- At the local location (if CURRENT SERVER is blank or specifies that location explicitly), the order is:
  a. All packages that are already allocated to the plan while the plan is running.
  b. All unallocated packages that are explicitly specified in, and all collections that are completely included in, the package list of the plan. Db2 searches for packages in the order that they appear in the package list.
- At a remote location, the order is:
  a. All packages that are already allocated to the plan at that location while the plan is running.
  b. All unallocated packages that are explicitly specified in, and all collections that are completely included in, the package list of the plan, whose locations match the value of CURRENT SERVER. Db2 searches for packages in the order that they appear in the package list.

  If you use the BIND PLAN option DEFER(PREPARE), Db2 does not search all collections in the package list.

If the order of search is not important

In many cases, the order in which Db2 searches the packages is not important to you and does not affect performance. For an application that runs only at your local Db2 system, you can name every package differently and include them all in the same collection. The package list on your BIND PLAN subcommand can read:

```
PKLIST (collection.*)
```

The resulting plan consists of the following information:

- Any programs that are associated with DBRMs in the MEMBER list
- Any programs that are associated with packages and collections that are identified in PKLIST

You can add packages to the collection even after binding the plan. Db2 lets you bind packages having the same package name into the same collection only if their version IDs are different.

If your application uses DRDA access, you must bind some packages at remote locations. Use the same collection name at each location, and identify your package list as:

```
PKLIST (*.collection.*)
```

If you use an asterisk for part of a name in a package list, Db2 checks the authorization for the package to which the name resolves at run time. To avoid the checking at run time in the preceding example, you can grant EXECUTE authority for the entire collection to the owner of the plan before you bind the plan.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Related reference

BIND PLAN (DSN) (Db2 Commands)

CURRENT PACKAGE PATH (Db2 SQL)

CURRENT PACKAGESET (Db2 SQL)

How Db2 identifies packages at run time

The Db2 precompiler or Db2 coprocessor identifies each call to Db2 with a consistency token. The same consistency token identifies the DBRM that the SQL statement processor produces and the package to which you bound the DBRM.

When you run the program, Db2 uses the consistency token in matching the call to Db2 to the correct DBRM. Usually, the consistency token is in an internal Db2 format. You can override that token if you want.

You also need other identifiers. The consistency token alone does not necessarily identify a unique package. You can bind the same DBRM to many packages, at different locations and in different collections, and you can include all those packages in the package list of the same plan. All those packages will have the same consistency token. You can specify a particular location or a particular collection at run time.

Related tasks

Setting the program level
The program level defines the level for a particular module. This information is stored in the consistency token, which is in an internal Db2 format. Overriding the program level in the consistency token is possible, if needed, but generally not recommended.

Specifying the location of the package that Db2 is to use

When your program executes SQL statements, Db2 uses the value in the CURRENT SERVER special register to determine the location of the necessary package. If the current server is your local Db2 subsystem and it does not have a location name, the value in the special register is blank.

About this task

You can change the value of CURRENT SERVER by using the SQL CONNECT statement in your program. If you do not use CONNECT, the value of CURRENT SERVER is the location name of your local Db2 subsystem (or blank, if your Db2 subsystem has no location name).

Specifying the package collection that Db2 is to use

To ensure that Db2 uses the intended package collection and does not waste time searching, explicitly specify the package collection that you want Db2 to use.

About this task

You can use the special register CURRENT PACKAGE PATH or CURRENT PACKAGESET (if CURRENT PACKAGE PATH is not set) to specify the collections that are to be used for package resolution. The CURRENT PACKAGESET special register contains the name of a single collection, and the CURRENT PACKAGE PATH special register contains a list of collection names.

If you do not set these registers, they contain an empty string when your application begins to run, and they remain as an empty string. In this case, Db2 searches the available collections.

However, explicitly specifying the intended collection by using the special registers can avoid a potentially costly search through a package list that has many qualifying entries. In addition, Db2 uses the values in these special registers for applications that do not run under a plan.

When you call a stored procedure, the special register CURRENT PACKAGESET contains the value that you specified for the COLLID parameter when you defined the stored procedure. If the routine was defined without a value for the COLLID parameter, the value of the special register is inherited from the calling program. Also, the special register CURRENT PACKAGE PATH contains the value that you specified for the PACKAGE PATH parameter when you defined the stored procedure. When the stored procedure returns control to the calling program, Db2 restores this register to the value that it contained before the call.

Specifying the package collection for a Java routine in a JAR file that is installed in the Db2 catalog: If the Java routine definition specifies a COLLID value that is different from the collection into which the IBM Data Server Driver for JDBC and SQLJ packages are bound, you need to run the DB2Binder utility with the -collection option to bind the driver packages into the collection that is specified by the COLLID value in the Java routine definition. If you do not do so, you might receive SQLCODE -805.

Related tasks

Binding an application plan
An application plan can include package lists.

Overriding the values that Db2 uses to resolve package lists
Db2 resolves package lists by searching the available collections in a particular order. To avoid this search, you can specify the values that Db2 should use for package resolution.

Related reference

DB2Binder utility (Db2 Application Programming for Java)

Overriding the values that Db2 uses to resolve package lists

Db2 resolves package lists by searching the available collections in a particular order. To avoid this search, you can specify the values that Db2 should use for package resolution.

About this task

If you set the special register CURRENT PACKAGE PATH or CURRENT PACKAGESET, Db2 skips the check for programs that are part of a plan and uses the values in these registers for package resolution.

If you set CURRENT PACKAGE PATH, Db2 uses the value of CURRENT PACKAGE PATH as the collection name list for package resolution. For example, if CURRENT PACKAGE PATH contains the list COLL1, COLL2, COLL3, COLL4, Db2 searches for the first package that exists in the following order:

COLL1.PROG1.*timestamp*
COLL2.PROG1.*timestamp*
COLL3.PROG1.*timestamp*
COLL4.PROG1.*timestamp*

If you set CURRENT PACKAGESET and **not** CURRENT PACKAGE PATH, Db2 uses the value of CURRENT PACKAGESET as the collection for package resolution. For example, if CURRENT PACKAGESET contains COLL5, Db2 uses COLL5.PROG1.*timestamp* for the package search.

When CURRENT PACKAGE PATH is set, the server that receives the request ignores the collection that is specified by the request and instead uses the value of CURRENT PACKAGE PATH at the server to resolve the package. Specifying a collection list with the CURRENT PACKAGE PATH special register can avoid the need to issue multiple SET CURRENT PACKAGESET statements to switch collections for the package search.

The following table shows examples of the relationship between the CURRENT PACKAGE PATH special register and the CURRENT PACKAGESET special register.

Table 142. Scope of CURRENT PACKAGE PATH

Example	What happens
`SET CURRENT PACKAGESET` `SELECT ... FROM T1 ...`	The collection in PACKAGESET determines which package is invoked.
`SET CURRENT PACKAGE PATH` `SELECT ... FROM T1 ...`	The collections in PACKAGE PATH determine which package is invoked.

`SET CURRENT PACKAGESET`
`SET CURRENT PACKAGE PATH`
`SELECT ... FROM T1 ...`
	The collections in PACKAGE PATH determine which package is invoked.

Table 142. Scope of CURRENT PACKAGE PATH (continued)

Example	What happens

`SET CURRENT PACKAGE PATH`
`CONNECT TO S2 ...`
`SELECT ... FROM T1 ...`
	PACKAGE PATH at server S2 is an empty string because it has not been explicitly set. The values from the PKLIST bind option of the plan that is at the requester determine which package is invoked.[1]

`SET CURRENT PACKAGE PATH`
`   = 'A,B'`
`CONNECT TO S2 ...`
`SET CURRENT PACKAGE PATH`
`   = 'X,Y'`
`SELECT ... FROM T1 ...`
	The collections in PACKAGE PATH that are set at server S2 determine which package is invoked.

`SET CURRENT PACKAGE PATH`
`SELECT ... FROM`
`S2.QUAL.T1 ...`
	Three-part table name. On implicit connection to server S2, PACKAGE PATH at server S2 is inherited from the local server. The collections in PACKAGE PATH at server S2 determine which package is invoked.

Notes:

1. When CURRENT PACKAGE PATH is set at the requester (and not at the remote server), Db2 passes one collection at a time from the list of collections to the remote server until a package is found or until the end of the list. Each time a package is not found at the server, Db2 returns an error to the requester. The requester then sends the next collection in the list to the remote server.

Bind process for remote access

You can use several different bind processes to enable access to data at a remote server.

These processes work when the remote server is a Db2 for z/OS system or another type of database system that uses DRDA access.

Bind a package at the local site and the remote site

If you have not yet bound a local package, use this technique.

1. Bind the DBRM into a package at the local site.
2. Bind the DBRM into a package at the remote site.
3. Bind a plan with a package list that includes the local package and the remote package.

Example: Bind a package at the local site and the remote site

Suppose that you precompiled program MYPROG to generate DBRM MYPROG, and compiled and link-edited program MYPROG. You want to run MYPROG to access tables at site CHICAGO from your local site. Use commands like these to bind local and remote packages and a plan. The number at the end of each line corresponds to a previously described step.

```
BIND PACKAGE(LOCALCOLLID) MEMBER(MYPROG) other bind options            1
BIND PACKAGE (CHICAGO.REMOTECOLLID) MEMBER(MYPROG) other bind options  2
BIND PLAN (MYPLAN) PKLIST(LOCALCOLLID.* ,*.REMOTECOLLID.*)             3
```

Bind a copy of an existing local package at the remote site

If you have already bound a local package, you can use this technique.

1. Issue the BIND command with COPY and OPTIONS to create a copy of the local package at the remote site.

   OPTIONS controls the option values for bind options that you do not specify.

2. Bind a plan with a package list that includes the local package and the remote package.

Example: Bind a copy of an existing local package at the remote site

Suppose that you previously prepared program MYPROG for execution at the local site. As part of program preparation, you bound package LOCALCOLLID.MYPROG at the local site. Now you want to run MYPROG to access tables at site CHICAGO from your local site. Use commands like these to bind a copy of the local package at site CHICAGO, and then bind a plan. The number at the end of each line corresponds to a previously described step.

```
BIND PACKAGE(CHICAGO.REMOTECOLLID) COPY(LOCALCOLLID.MYPROG) -
 OPTIONS(COMPOSITE|COMMAND) -
 other bind options                                                    1
BIND PLAN (MYPLAN) PKLIST(LOCALCOLLID.* ,*.REMOTECOLLID.*)             2
```

Bind options for remote access

Binding a package to run at a remote location is like binding a package to run at your local Db2 subsystem. Binding a plan to run the package is like binding any other plan. However, a few differences exist.

For the general instructions, see Chapter 9, "Preparing an application to run on Db2 for z/OS," on page 825.

BIND PLAN options for DRDA access

The following options of BIND PLAN are particularly relevant to binding a plan that uses DRDA access:

DISCONNECT

For most flexibility, use DISCONNECT(EXPLICIT), explicitly or by default. That requires you to use RELEASE statements in your program to explicitly end connections.

The other values of the option are also useful.

DISCONNECT(AUTOMATIC) ends all remote connections during a commit operation, without the need for RELEASE statements in your program.

DISCONNECT(CONDITIONAL) ends remote connections during a commit operation except when an open cursor defined as WITH HOLD is associated with the connection.

SQLRULES

Use **SQLRULES(Db2)**, explicitly or by default.

SQLRULES(STD) applies the rules of the SQL standard to your CONNECT statements, so that CONNECT TO *x* is an error if you are already connected to *x*. Use STD only if you want that statement to return an error code.

If your program selects LOB data from a remote location, and you bind the plan for the program with SQLRULES(Db2), the format in which you retrieve the LOB data with a cursor is restricted. After you open the cursor to retrieve the LOB data, you must retrieve all of the data using a LOB variable, or retrieve all of the data using a LOB locator variable. If the value of SQLRULES is STD, this restriction does not exist.

If you intend to switch between LOB variables and LOB locators to retrieve data from a cursor, execute the SET SQLRULES=STD statement before you connect to the remote location.

CURRENTDATA

Use **CURRENTDATA(NO)** to force block fetch for ambiguous cursors.

ENCODING

Use this option to control the encoding scheme that is used for static SQL statements in the plan and to set the initial value of the CURRENT APPLICATION ENCODING SCHEME special register.

For applications that execute remotely and use explicit CONNECT statements, Db2 uses the ENCODING value for the plan. For applications that execute remotely and use implicit CONNECT statements, Db2 uses the ENCODING value for the package that is at the site where a statement executes.

BIND PACKAGE options for DRDA access

The following options of BIND PACKAGE are relevant to binding a package to be run using DRDA access:

location-name
Name the location of the server at which the package runs.

The privileges needed to run the package must be granted to the owner of the package at the server. If you are not the owner, you must also have SYSCTRL authority or the BINDAGENT privilege that is granted locally.

SQLERROR
Use **SQLERROR(CONTINUE)** if you used SQL(ALL) when precompiling. That creates a package even if the bind process finds SQL errors, such as statements that are valid on the remote server but that the precompiler did not recognize. Otherwise, use SQLERROR(NOPACKAGE), explicitly or by default.

COPY
If you bind with the COPY option to copy a local package to a remote site, Db2 performs authorization checking, reads and updates the catalog, and creates the package at the remote site. Db2 reads the catalog records that are related to the copied package at the local site. Db2 converts values that are returned from the remote site in ISO format if all of the following conditions are true:

- If the local site is installed with time or date format LOCAL
- A package is created at a remote site with the COPY option
- The SQL statement does not specify a different format.

CURRENTDATA
Use **CURRENTDATA(NO)** to force block fetch for ambiguous cursors.

OPTIONS
When you make a remote copy of a package using BIND PACKAGE with the COPY option, use this option to control the default bind options that Db2 uses. Specify:

COMPOSITE to cause Db2 to use any options you specify in the BIND PACKAGE command. For all other options, Db2 uses the options of the copied package. COMPOSITE is the default.

COMMAND to cause Db2 to use the options you specify in the BIND PACKAGE command. For all other options, Db2 uses the defaults for the server on which the package is bound. This helps ensure that the server supports the options with which the package is bound.

ENCODING
Use this option to control the encoding scheme that is used for static SQL statements in the package and to set the initial value of the CURRENT APPLICATION ENCODING SCHEME special register.

When you bind the same package locally and remotely, and you specify the ENCODING bind option for the package, the ENCODING bind option for the local package applies to the remote application. The default ENCODING value for a package that is bound at a remote Db2 for z/OS server is the system default for that server. The system default is specified at installation time in the APPLICATION ENCODING field of panel DSNTIPF, which is the APPENSCH DECP value.

EXPLAIN
If you specify the option EXPLAIN(YES) or EXPLAIN(ONLY), and you do not specify the option SQLERROR(CONTINUE), PLAN_TABLE must exist at the remote location at which the package is bound.

Related concepts

Bind options for locks (Db2 Performance)

Related tasks

BIND options for distributed applications (Db2 Performance)

Related reference
BIND and REBIND options for packages, plans, and services (Db2 Commands)

Considerations for binding packages at a remote location

When you bind packages at a remote location, you need to understand how the behavior of the remote packages differs from the behavior of local packages.

Scope of a remote bind

When you bind or rebind a package at a remote site, Db2 checks authorizations, reads and updates the catalog, and creates the package in the directory at the remote site. Db2 does not read or update catalogs or check authorizations at the local site.

After you bind a remote package, you can bind, rebind, or free the remote package from the local site or at the remote site.

Authorization for binding and running packages at a remote location

To bind a package at a remote Db2 system, you must have all the privileges or authority there that you would need to bind the package on your local system. To bind a package at another type of a system, such as Db2 for Linux, UNIX, and Windows, you need all privileges that the other system requires to execute the SQL statements in the package, and to access the data objects to which the package refers.

Communications database requirements for binding packages at a remote location

When you bind a package at a remote site, the local communications database must be able to resolve the location name in the package to a remote location.

Remote access through a stored procedure

If a local stored procedure uses a cursor to access data, and the cursor-related statement is bound in a separate package under the stored procedure, you must bind this separate package both locally and remotely. In addition, the invoker or owner of the stored procedure must be authorized to execute both local and remote packages. At your local requesting system, you must bind a plan whose package list includes all of those local and remote packages.

Checking which BIND PACKAGE options a particular server supports

You can request only the options of the BIND PACKAGE command that are supported by the server by specifying those options at the requester.

About this task

To find out which options are supported by a specific server DBMS, refer to the documentation provided for that server.

For specific Db2 bind information, refer to the following documentation:

- For guidance in using Db2 bind options and performing a bind process, see Chapter 9, “Preparing an application to run on Db2 for z/OS,” on page 825.
- For the syntax of Db2 BIND command, see the topics BIND PACKAGE (DSN) (Db2 Commands) and BIND PLAN (DSN) (Db2 Commands).
- For the syntax of Db2 REBIND command, see the topics REBIND PACKAGE (DSN) (Db2 Commands) and REBIND PLAN (DSN) (Db2 Commands).

Binding a batch program

Before a batch program can issue SQL statements, a Db2 plan must exist.

About this task

The owner of the plan or package must have all the privileges that are required to execute the SQL statements embedded in it.

You can specify the plan name to Db2 in one of the following ways:

- In the DDITV02 input data set.
- In subsystem member specification.
- By default; the plan name is then the application load module name that is specified in DDITV02.

Db2 passes the plan name to the IMS attach package. If you do not specify a plan name in DDITV02, and a resource translation table (RTT) does not exist or the name is not in the RTT, Db2 uses the passed name as the plan name. If the name exists in the RTT, the name translates to the plan that is specified for the RTT.

Recommendation: Give the Db2 plan the same name as that of the application load module, which is the IMS attachment facility default. The plan name must be the same as the program name.

Conversion of DBRMs that are bound to a plan to DBRMs that are bound to a package

You must bind all DBRMs into a package, and bind the packages into a plan. One package can have only one DBRM.

The default REBIND PLAN COLLID (*) option converts all plans with DBRMs into plans with a package list. You can use this technique for local applications only. If the plan that you specify already contains both DBRMs and package lists, the newly converted package entries will be inserted into the front of the existing package list.

Important: If the same DBRM is in multiple plans, and you run REBIND PLAN with the same COLLID option value on more than one of those plans, Db2 overlays the previously created package in the collection each time you run REBIND with COLLID. To avoid overlaying packages, specify a different COLLID value for each plan that contains DBRMs that are also in other plans.

For more information on developing a strategy for converting your plans to include only packages, see DB2 9 for z/OS: Packages Revisited (IBM Redbooks).

Example: converting all plans

The following examples converts all DBRMs that are bound with plan X into packages under collection ID: DSN_DEFAULT_COLLID_X.

```
REBIND PLAN(X) COLLID(*);
```

Example: specifying a collection ID

The following examples converts DBRMs that are bound with plan X into packages under the *my_collection* collection ID.

```
REBIND PLAN(x) COLLID('my_collection');
```

Example: rebinding multiple plans which may contain DBRMs

In the following example, BIND will traverse through each plan that is specified in the REBIND PLAN command statement and will convert the DBRMs accordingly, and until none of the DBRMs are bound with plans.

```
REBIND PLAN (X1, X2, X3) COLLID (collection_id|*);
```

Example: rebinding all plans which may contain DBRMs

In the following example, BIND will traverse through all plans that are specified in the SYSPLAN table and will convert the DBRMs accordingly, and until none of the DBRMs are bound with plans.

```
REBIND PLAN (*) COLLID (collection_id|*);
```

Example: specifying a package list

The following examples converts all DBRMs that are bound with plan X into packages under collection ID: DSN_DEFAULT_COLLID_X.

- If plan X does not have a package list, the newly converted package entries will be appended to the front of package list Z and then package list Z will be added to plan X.
- If plan X has both a package list and DBRMs, the newly converted package entries will be appended to the front of package list Z and then package list Z will replace the existing package list.
- If plan X has only a package list, then package list Z will replace the existing package list.

```
REBIND PLAN (x) COLLID (collection_id|*) PKLIST(Z);
```

Example: specifying no package list

The following examples converts all DBRMs that are bound with plan X into packages under collection ID: DSN_DEFAULT_COLLID_X.

- If plan X has both a package list and DBRMs, the existing package list will be deleted, and the new package list will be bound into plan X.
- If plan X has only DBRMs, the DBRMs will be converted into packages accordingly and added to plan X. The NOPKLIST option will be ignored.
- If plan X does not have DBRMs, then the existing package list, if any, will be deleted.

```
REBIND PLAN (x) COLLID (collection_id|*) NOPKLIST;
```

Converting an existing plan into packages to run remotely

If you have an existing application that you want to run at a remote location by using remote access, you need a new plan that includes those remote packages in its package list.

Procedure

To turn an existing plan with member DBRMs into packages to run remotely, perform the following actions for each remote location:

1. Choose a name for a collection to contain member DBRMs, such as REMOTE1.
2. Convert the plan into a plan with a package list of packages.

   ```
   REBIND PLAN(REMOTE1)COLLID(*)
   ```

   Specifying COLLID(*) produces the packages under the collection of DSN_DEFAULT_COLLID_planname.

3. Query SYSIBM.SYSPACKDEP, to see if any of the packages have a dependency on an alias. That alias is a definition for a 3-part name.

   a) For each of the packages that have a dependency on an alias:

   ```
   BIND PACKAGE(location.remote_server_collid)
    COPY(DSN_DEFAULT_COLLID_planname.pgkid)
    COPYVER(...) OPTIONS(COMPOSITE)
   ```

4. Adjust the location's package list. If prior to this process, the plan had no package list, after "2" on page 866 it will have a package list containing DSN_DEFAULT_COLLID_*planname.pgkid*.

   ```
   REBIND PLAN PKLIST
    (*.DSN_DEFAULT_COLLID_planname.pgkid* *.remote_server_collid.* )
   ```

Results

When you now run the existing application at your local Db2 system using the new application plan, these things happen:

- You connect immediately to the remote location that is named in the CURRENTSERVER option.
- Db2 searches for the package in the collection REMOTE1 at the remote location.
- Any UPDATE, DELETE, or INSERT statements in your application affect tables at the remote location.
- Any results from SELECT statements are returned to your existing application program, which processes them as though they came from your local Db2 system.

Setting the program level

The program level defines the level for a particular module. This information is stored in the consistency token, which is in an internal Db2 format. Overriding the program level in the consistency token is possible, if needed, but generally not recommended.

Procedure

Use the LEVEL (*aaaa*) option.

Db2 uses the value that you choose for *aaaa* to generate the consistency token. Although this method is not recommended for general use and the DSNH CLIST or the Db2 Program Preparation panels do not support it, this method enables you to perform the following actions:

a. Change the source code (but not the SQL statements) in the Db2 precompiler output of a bound program.
b. Compile and link-edit the changed program.
c. Run the application without rebinding a plan or package.

Dynamic rules options for dynamic SQL statements

The DYNAMICRULES bind option and the runtime environment determine the rules for the dynamic SQL attributes.

The BIND or REBIND option DYNAMICRULES determines what values apply at run time for the following dynamic SQL attributes:

- The authorization ID that is used to check authorization
- The qualifier that is used for unqualified objects
- The source for application programming options that Db2 uses to parse and semantically verify dynamic SQL statements
- Whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements

In addition, the runtime environment of a package controls how dynamic SQL statements behave at run time. The two possible runtime environments are:

- The package runs as part of a stand-alone program.
- The package runs as a stored procedure or user-defined function package, or it runs under a stored procedure or user-defined function.

  A package that runs under a stored procedure or user-defined function is a package whose associated program meets one of the following conditions:

  – The program is called by a stored procedure or user-defined function.

  – The program is in a series of nested calls that start with a stored procedure or user-defined function.

Dynamic SQL statement behavior

The dynamic SQL attributes that are determined by the value of the DYNAMICRULES bind option and the runtime environment are collectively called the *dynamic SQL statement behavior* or the *dynamic rules behavior*. The four dynamic rules behaviors are: run, bind, define, and invoke.

The following table shows the combination of DYNAMICRULES value and runtime environment that yield each dynamic SQL behavior.

Table 143. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior

	Behavior of dynamic SQL statements	
DYNAMICRULES value	**Stand-alone program environment**	**User-defined function or stored procedure environment**
RUN	Run	Run
BIND	Bind	Bind
DEFINERUN	Run	Define
DEFINEBIND	Bind	Define
INVOKERUN	Run	Invoke
INVOKEBIND	Bind	Invoke

Note: BIND and RUN values can be specified for packages, plans, and native SQL procedures. The other values can be specified for packages and native SQL procedures but not for plans.

The following table shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

Table 144. Definitions of dynamic SQL statement behaviors

Dynamic SQL attribute	**Setting for dynamic SQL behavior attributes**			
	Bind	**Run**	**Define**	**Invoke**
Authorization ID	Plan or package owner	Current SQLID	User-defined function or stored procedure owner	Authorization ID of invoker[1]
Default qualifier for unqualified objects	Bind OWNER or QUALIFIER value	CURRENT SCHEMA	User-defined function or stored procedure owner	Authorization ID of invoker
CURRENT SQLID[2]	Not applicable	Applies	Not applicable	Not applicable

Table 144. Definitions of dynamic SQL statement behaviors (continued)

Dynamic SQL attribute	Setting for dynamic SQL behavior attributes			
	Bind	**Run**	**Define**	**Invoke**
Source for application programming options	Determined by DSNHDECP or a user-specified application defaults module parameter DYNRULS[3]	Install panel DSNTIP4	Determined by DSNHDECP or a user-specified application defaults module parameter DYNRULS[3]	Determined by DSNHDECP or a user-specified application defaults module parameter DYNRULS[3]
Can execute GRANT, REVOKE, CREATE, ALTER, DROP, RENAME?	No	Yes	No	No

Notes:

1. If the invoker is the primary authorization ID of the process or the CURRENT SQLID value, secondary authorization IDs are also checked if they are needed for the required authorization. Otherwise, only one ID, the ID of the invoker, is checked for the required authorization.
2. Db2 uses the value of CURRENT SQLID as the authorization ID for dynamic SQL statements only for plans and packages that have run behavior. For the other dynamic SQL behaviors, Db2 uses the authorization ID that is associated with each dynamic SQL behavior, as shown in this table.

   The value to which CURRENT SQLID is initialized is independent of the dynamic SQL behavior. For stand-alone programs, CURRENT SQLID is initialized to the primary authorization ID.

   You can execute the SET CURRENT SQLID statement to change the value of CURRENT SQLID for packages with any dynamic SQL behavior, but Db2 uses the CURRENT SQLID value only for plans and packages with run behavior.
3. The value of DSNHDECP or a user-specified application defaults module parameter DYNRULS, which you specify in field USE FOR DYNAMICRULES in installation panel DSNTIP4, determines whether Db2 uses the SQL statement processing options or the application programming defaults for dynamic SQL statements. See "Options for SQL statement processing" on page 841 for more information.

Related concepts

Authorization IDs and dynamic SQL (Db2 SQL)

Authorization behaviors for dynamic SQL statements (Managing Security)

Related reference

DYNAMICRULES bind option (Db2 Commands)

Dynamic plan selection

It is beneficial to use dynamic plan selection and packages together. You can convert individual programs in an application that contains many programs and plans, one at a time, to use a combination of plans and packages. This process reduces the number of plans per application; having fewer plans reduces the effort that is needed to maintain the dynamic plan exit routine.

CICSYou can use packages and dynamic plan selection together, but when you dynamically switch plans, the following conditions must exist:

- All special registers, including CURRENT PACKAGESET, must contain their initial values.
- The value in the CURRENT DEGREE special register cannot have changed during the current transaction.

Assume that you develop the following programs and DBRMs:

Table 145. Example programs and DBRMs

Program Name	DBRM Name
MAIN	MAIN
PROGA	PLANA
PROGB	PKGB
PROGC	PLANC

You could create packages using the following bind statement:

```
BIND PACKAGE(PKGB) MEMBER(PKGB)
```

The following scenario illustrates thread association for a task that runs program MAIN. Suppose that you execute the following SQL statements in the indicated order. For each SQL statement, the resulting event is described.

1. `EXEC CICS START TRANSID(MAIN)`

   `TRANSID(MAIN)` executes program MAIN.
2. `EXEC SQL SELECT…`

   Program MAIN issues an SQL SELECT statement. The default dynamic plan exit routine selects plan MAIN.
3. `EXEC CICS LINK PROGRAM(PROGA)`

   Program PROGA is invoked.
4. `EXEC SQL SELECT…`

   Db2 does not call the default dynamic plan exit routine, because the program does not issue a sync point. The plan is MAIN.
5. `EXEC CICS LINK PROGRAM(PROGB)`

   Program PROGB is invoked.
6. `EXEC SQL SELECT…`

   Db2 does not call the default dynamic plan exit routine, because the program does not issue a sync point. The plan is MAIN and the program uses package PKGB.
7. `EXEC CICS SYNCPOINT`

   Db2 calls the dynamic plan exit routine when the next SQL statement executes.
8. `EXEC CICS LINK PROGRAM(PROGC)`

   Program PROGC is invoked.
9. `EXEC SQL SELECT…`

   Db2 calls the default dynamic plan exit routine and selects PLANC.
10. `EXEC SQL SET CURRENT SQLID = 'ABC'`

    The CURRENT SQLID special register is assigned the value 'ABC.'
11. `EXEC CICS SYNCPOINT`

    Db2 does not call the dynamic plan exit routine when the next SQL statement executes because the previous statement modifies the special register CURRENT SQLID.
12. `EXEC CICS RETURN`

    Control returns to program PROGB.
13. `EXEC SQL SELECT…`

CICS With packages, you probably do not need dynamic plan selection and its accompanying exit routine. A package that is listed within a plan is not accessed until it is executed. However, you can use dynamic plan selection and packages together, which can reduce the number of plans in an application and the effort to maintain the dynamic plan exit routine.

Rebinding applications

You must rebind applications to change bind options. You also need to rebind applications when you make changes that affect the plan or package, such as creating an index, but you have not changed the SQL statements.

About this task

In some cases, Db2 automatically rebinds plans or packages for you, depending on the value of the ABIND subsystem parameter. For details, see "Automatic rebinds" on page 879.

The following actions might require that you rebind a package:

- Changing the host language or SQL statements in the application. You must replace the package. Precompile, compile, and link the application program. Then issue a BIND command with the ACTION(REPLACE) option.
- Changing your data attributes in ways that invalidate the package. For details, see "Changes that invalidate packages" on page 17.
- Improving access paths selection after reorganizing data with the REORG utility or collecting database statistics with RUNSTATS or other utilities. For more information, see Maintaining data organization and statistics (Db2 Performance).
- Enabling Db2 to select an access path that uses a newly created index for access to a table.
- Changing the bind options for a package. If an option that you want to change is not available for the REBIND command, issue the BIND command with ACTION(REPLACE) instead.
- Preparing for migration to a new Db2 release. For more information, see Rebind old plans and packages in Db2 10 to avoid disruptive autobinds in Db2 11 (Db2 Installation and Migration).

Related tasks

Identifying packages with characteristics that affect performance, concurrency, or the ability to run (Db2 Performance)

Related reference

AUTO BIND field (ABIND subsystem parameter) (Db2 Installation and Migration)

REBIND PACKAGE (DSN) (Db2 Commands)

BIND PACKAGE (DSN) (Db2 Commands)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Rebinding a package

You need to rebind a package when you make changes that affect the package but that do not involve changes to the SQL statements. For example, if you create a new index, you need to rebind the package. If you change the SQL, you need to use the BIND PACKAGE command with the ACTION(REPLACE) option.

Before you begin

For a trigger package, use the REBIND TRIGGER PACAKGE subcommand. For more information, see "Trigger packages" on page 161.

Procedure

Use the REBIND PACKAGE subcommand.

You can change any of bind options for a package when you rebind it.

The following table clarifies which packages are bound, depending on how you specify *collection-id* (coll-id), *package-id* (pkg-id), and *version-id* (ver-id) on the REBIND PACKAGE subcommand.

REBIND PACKAGE does not apply to packages for which you do not have the BIND privilege. An asterisk (*) used as an identifier for collections, packages, or versions does not apply to packages at remote sites.

Table 146. Behavior of REBIND PACKAGE specification. "All" means all collections, packages, or versions at the local Db2 server for which the authorization ID that issues the command has the BIND privilege.

Input	Collections affected	Packages affected	Versions affected
`*`	all	all	all
`*.*.(*)`	all	all	all
`*.*`	all	all	all
`*.*.(ver-id)`	all	all	ver-id
`*.*.()`	all	all	empty string
`coll-id.*`	coll-id	all	all
`coll-id.*.(*)`	coll-id	all	all
`coll-id.*.(ver-id)`	coll-id	all	ver-id
`coll-id.*.()`	coll-id	all	empty string
`coll-id.pkg-id.(*)`	coll-id	pkg-id	all
`coll-id.pkg-id`	coll-id	pkg-id	empty string
`coll-id.pkg-id.()`	coll-id	pkg-id	empty string
`coll-id.pkg-id.(ver-id)`	coll-id	pkg-id	ver-id
`*.pkg-id.(*)`	all	pkg-id	all
`*.pkg-id`	all	pkg-id	empty string
`*.pkg-id.()`	all	pkg-id	empty string
`*.pkg-id.(ver-id)`	all	pkg-id	ver-id

Examples

Example: Rebinding a package at a remote location

The following example shows the options for rebinding a package at the remote location. The location name is SNTERSA. The collection is GROUP1, the package ID is PROGA, and the version ID is V1. The connection types shown in the REBIND subcommand replace connection types that are specified on the original BIND subcommand.

```
REBIND PACKAGE(SNTERSA.GROUP1.PROGA.(V1)) ENABLE(CICS,REMOTE)
```

Example: Rebinding all local packages

You can use the asterisk on the REBIND subcommand for local packages, but not for packages at remote sites. Any of the following commands rebinds all versions of all packages in all collections, at the local Db2 system, for which you have the BIND privilege.

```
REBIND PACKAGE (*)
REBIND PACKAGE (*.*)
REBIND PACKAGE (*.*.(*))
```

Example: Rebinding all versions of all local packages

Either of the following commands rebinds all versions of all packages in the local collection LEDGER for which you have the BIND privilege.

```
REBIND PACKAGE (LEDGER.*)
```

```
REBIND PACKAGE (LEDGER.*.(*))
```

Example: Rebinding local packages in all collections

Either of the following commands rebinds the empty string version of the package DEBIT in all collections, at the local Db2 system, for which you have the BIND privilege.

```
REBIND PACKAGE (*.DEBIT)
```

```
REBIND PACKAGE (*.DEBIT.())
```

Related tasks

Reusing and comparing access paths at bind and rebind (Db2 Performance)

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

REBIND PACKAGE (DSN) (Db2 Commands)

Rebinding a plan

You need to rebind a plan when you make a change to one of the attributes of the plan, such as the package list.

Procedure

Use the REBIND PLAN subcommand.

You can change any of bind options for that plan.

When you rebind a plan, use the PKLIST keyword to replace any previously specified package list. Omit the PKLIST keyword to use of the previous package list for rebinding. Use the NOPKLIST keyword to delete any package list that was specified when the plan was previously bound.

Examples

Example

The following command rebinds PLANA and changes the package list:

```
REBIND PLAN(PLANA) PKLIST(GROUP1.*) MEMBER(ABC)
```

Example

The following command rebinds the plan and drops the entire package list:

```
REBIND PLAN(PLANA) NOPKLIST
```

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)
REBIND PLAN (DSN) (Db2 Commands)

Rebinding lists of plans and packages

In some situations, you need to rebind a set of plans or packages that cannot be described by using asterisks. For example, if a rebind operation terminates, you can generate a rebind subcommand for each object that was not bound.

About this task

One situation in which this technique is useful is to complete a rebind operation that has terminated due to lack of resources. A rebind for many objects, such as REBIND PACKAGE (*) for an ID with SYSADM authority, terminates if a needed resource becomes unavailable. As a result, some objects are successfully rebound and others are not. If you repeat the subcommand, Db2 attempts to rebind all the objects again. But if you generate a rebind subcommand for each object that was not rebound, and issue those subcommands, Db2 does not repeat any work that was already done and is not likely to run out of resources.

For a description of the technique and several examples of its use, see "Sample program to create REBIND subcommands for lists of plans and packages" on page 874.

Generating lists of REBIND commands

To generate a list of REBIND subcommands for a set of packages that cannot be described, use asterisks, and use information in the Db2 catalog. You can then issue the list of subcommands through DSN.

About this task

The following list is an overview of the procedures for REBIND PACKAGE:

1. Use DSNTIAUL to generate the REBIND PACKAGE subcommands for the selected packages.
2. Use DSNTEDIT CLIST to delete extraneous blanks from the REBIND PACKAGE subcommands.
3. Use TSO edit commands to add DSN commands to the sequential data set.
4. Use DSN to execute the REBIND PACKAGE subcommands for the selected packages.

Sample program to create REBIND subcommands for lists of plans and packages

If you cannot use asterisks to identify a list of packages or plans that you want to rebind, you might be able to create the needed REBIND subcommands automatically, by using the sample program DSNTIAUL.

One situation in which this technique might be useful is when a resource becomes unavailable during a rebind of many plans or packages. Db2 normally terminates the rebind and does not rebind the remaining plans or packages. Later, however, you might want to rebind only the objects that remain to be rebound. You can build REBIND subcommands for the remaining plans or packages by using DSNTIAUL to select the plans or packages from the Db2 catalog and to create the REBIND subcommands. You can then submit the subcommands through the DSN command processor, as usual.

You might first need to edit the output from DSNTIAUL so that DSN can accept it as input. The CLIST DSNTEDIT can perform much of that task for you.

This section contains the following topics:

Sample SELECT statements for generating REBIND commands

You can select specific plans or packages to be rebound and concatenate the REBIND subcommand syntax around the plan or package names. You can also convert a varying-length string to a fixed-length string, and append additional blanks to the REBIND PLAN and REBIND PACKAGE subcommands, so that the DSN command processor can accept the record length as valid input.

Building REBIND subcommands: The examples that follow illustrate the following techniques:

- Using SELECT to select specific packages or plans to be rebound
- Using the CONCAT operator to concatenate the REBIND subcommand syntax around the plan or package names
- Using the SUBSTR function to convert a varying-length string to a fixed-length string
- Appending additional blanks to the REBIND PLAN and REBIND PACKAGE subcommands, so that the DSN command processor can accept the record length as valid input

If the SELECT statement returns rows, then DSNTIAUL generates REBIND subcommands for the plans or packages identified in the returned rows. Put those subcommands in a sequential data set, where you can then edit them.

For REBIND PACKAGE subcommands, delete any extraneous blanks in the package name, using either TSO edit commands or the Db2 CLIST DSNTEDIT.

For both REBIND PLAN and REBIND PACKAGE subcommands, add the DSN command that the statement needs as the first line in the sequential data set, and add END as the last line, using TSO edit commands. When you have edited the sequential data set, you can run it to rebind the selected plans or packages.

If the SELECT statement returns no qualifying rows, then DSNTIAUL does not generate REBIND subcommands.

The examples in this topic generate REBIND subcommands that work in Db2 for z/OS Db2 11. You might need to modify the examples for prior releases of Db2 that do not allow all of the same syntax.

Example: REBIND all plans without terminating because of unavailable resources.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT')                         ',1,45)
FROM SYSIBM.SYSPLAN;
```

Example: REBIND all versions of all packages without terminating because of unavailable resources.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))         ',1,55)
FROM SYSIBM.SYSPACKAGE;
```

Example: REBIND all plans bound before a given date and time.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT')                         ',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE <= 'yymmdd' OR
     (BINDDATE <= 'yymmdd' AND
      BINDTIME <= 'hhmmssth');
```

where *yymmdd* represents the date portion and *hhmmssth* represents the time portion of the timestamp string.

If the date specified is after 2000, you need to include another condition that includes plans that were bound before year 2000:

```
WHERE
      BINDDATE >= '830101' OR
      BINDDATE <= 'yymmdd' OR
     (BINDDATE <= 'yymmdd' AND
      BINDTIME <= 'hhmmssth');
```

Example: REBIND all versions of all packages bound before a given date and time.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))        ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME <= 'timestamp';
```

where *timestamp* is an ISO timestamp string.

Example: REBIND all plans bound since a given date and time.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT')                        ',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE >= 'yymmdd' AND
  BINDTIME >= 'hhmmssth';
```

where *yymmdd* represents the date portion and *hhmmssth* represents the time portion of the timestamp string.

Example: REBIND all versions of all packages bound since a given date and time.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID
  CONCAT'.'CONCAT NAME
  CONCAT'.(*))        ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME >= 'timestamp';
```

where *timestamp* is an ISO timestamp string.

Example: REBIND all plans bound within a given date and time range.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT')                        ',1,45)
FROM SYSIBM.SYSPLAN
WHERE
 (BINDDATE >= 'yymmdd' AND
  BINDTIME >= 'hhmmssth') AND
  BINDDATE <= 'yymmdd' AND
  BINDTIME <= 'hhmmssth');
```

where *yymmdd* represents the date portion and *hhmmssth* represents the time portion of the timestamp string.

Example: REBIND all versions of all packages bound within a given date and time range.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))        ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME >= 'timestamp1' AND
  BINDTIME <= 'timestamp2';
```

where *timestamp1* and *timestamp2* are ISO timestamp strings.

Example: REBIND all invalid versions of all packages.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))        ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE VALID = 'N';
```

Example: REBIND all plans bound with ISOLATION level of cursor stability.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT')                        ',1,45)
FROM SYSIBM.SYSPLAN
WHERE ISOLATION = 'S';
```

Example: REBIND all versions of all packages that allow CPU and/or I/O parallelism.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))        ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE DEGREE='ANY';
```

Sample JCL for running lists of REBIND commands

You can use JCL to rebind all versions of all packages that are bound within a specified date and time period.

You specify the date and time period for which you want packages to be rebound in a WHERE clause of the SELECT statement that contains the REBIND command. In The following example, the WHERE clause looks like the following clause:

```
WHERE BINDTIME >= 'YYYY-MM-DD-hh.mm.ss' AND
              BINDTIME <= 'YYYY-MM-DD-hh.mm.ss'
```

The date and time period has the following format:

YYYY
The four-digit year. For example: 2008.

MM
The two-digit month, which can be a value between 01 and 12.

DD
The two-digit day, which can be a value between 01 and 31.

hh
The two-digit hour, which can be a value between 01 and 24.

mm
The two-digit minute, which can be a value between 00 and 59.

ss
The two-digit second, which can be a value between 00 and 59.

```
//REBINDS  JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
//             REGION=1024K
//*********************************************************************/
//SETUP    EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIBB1) PARMS('SQL') -
      LIB('DSN1110.RUNLIB.LOAD')
 END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
//            UNIT=SYSDA,DISP=SHR
```

```
//*********************************************************************/
//*
//* GENER= '<SUBCOMMANDS TO REBIND ALL PACKAGES BOUND IN YYYY
//*
//*********************************************************************/
//SYSIN    DD *
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
  CONCAT NAME CONCAT'.(*))        ',1,55)
   FROM SYSIBM.SYSPACKAGE
   WHERE BINDTIME >= 'YYYY-MM-DD-hh.mm.ss' AND
```

```
              BINDTIME <= 'YYYY-MM-DD-hh.mm.ss';
/*
//*********************************************************************/
//*
//* STRIP THE BLANKS OUT OF THE REBIND SUBCOMMANDS
//*
//*********************************************************************/
//STRIP    EXEC PGM=IKJEFT01
//SYSPROC  DD DSN=SYSADM.DSNCLIST,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD *
 DSNTEDIT SYSADM.SYSTSIN.DATA
//SYSIN    DD DUMMY
/*
//*********************************************************************/
//*
//* PUT IN THE DSN COMMAND STATEMENTS
//*
//*********************************************************************/
//EDIT     EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
 TOP
 INSERT DSN SYSTEM(DSN)
 BOTTOM
 INSERT END
 TOP
 LIST * 99999
 END SAVE
/*
```

```
//*********************************************************************/
//*
//* EXECUTE THE REBIND PACKAGE SUBCOMMANDS THROUGH DSN
//*
//*********************************************************************/
//LOCAL    EXEC PGM=IKJEFT01
//DBRMLIB  DD DSN=DSN1110.DBRMLIB.DATA,
//            DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD DSN=SYSADM.SYSTSIN.DATA,
//            UNIT=SYSDA,DISP=SHR
/*
```

The following example shows some sample JCL for rebinding all plans bound without specifying the DEGREE keyword on BIND with DEGREE(ANY).

```
//REBINDS  JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
//             REGION=1024K
//*********************************************************************/
//SETUP    EXEC TSOBATCH
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
//            UNIT=SYSDA,DISP=SHR
//*********************************************************************/
//*
//* REBIND ALL PLANS THAT WERE BOUND WITHOUT SPECIFYING THE DEGREE
//* KEYWORD ON BIND WITH DEGREE(ANY)
//*
//*********************************************************************/
//SYSTSIN DD *
 DSN S(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIBB1) PARM('SQL')
 END
//SYSIN DD *
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
  CONCAT') DEGREE(ANY)            ',1,45)
   FROM SYSIBM.SYSPLAN
   WHERE DEGREE = ' ';
/*
//*********************************************************************/
//*
//* PUT IN THE DSN COMMAND STATEMENTS
```

```
//*
//*****************************************************************/
//EDIT    EXEC PGM=IKJEFT01
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
 TOP
 INSERT DSN S(DSN)
 BOTTOM
 INSERT END
 TOP
 LIST * 99999
 END SAVE
/*
//*****************************************************************/
//*
//* EXECUTE THE REBIND SUBCOMMANDS THROUGH DSN
//*
//*****************************************************************/
//REBIND  EXEC PGM=IKJEFT01
//STEPLIB  DD DSN=SYSADM.TESTLIB,DISP=SHR
//         DD DSN=DSN1110.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=SYSADM.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD DSN=SYSADM.SYSTSIN.DATA,DISP=SHR
//SYSIN    DD DUMMY
/*
```

Automatic rebinds

Automatic rebinds (sometimes called "autobinds") occur when an authorized user runs a package or plan and the runtime structures in the plan or package cannot be used. This situation usually results from changes to the attributes of the data on which the package or plan depends, or changes to the environment in which the package or plan runs.

In most cases, Db2 marks a package that must be automatically rebound as *invalid* by setting VALID='N' in the SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE catalog tables. For a list of actions that results in Db2 marking a package invalid, see "Changes that invalidate packages" on page 17.

If an automatic rebind fails, Db2 marks a package as *inoperative* in the OPERATIVE column of SYSIBM.SYSPLAN and SYSIBM.SYSPACKAGE catalog tables.

Controls for automatic binds

Db2 uses automatic binds only when the ABIND subsystem parameter is set to YES or COEXIST. If ABIND is set to NO when an invalid package runs, Db2 returns an error. For details, see AUTO BIND field (ABIND subsystem parameter) (Db2 Installation and Migration).

You can also use resource limit tables to control automatic binds. For details, see Restricting bind operations (Db2 Performance).

Bind options for automatic binds

In general, Db2 uses the same bind options from the most recent bind process for automatic binds. The exceptions are:

- If an option is no longer supported, the automatic rebind option process substitutes a supported option.
- If an option does not have an existing value, the default bind option is used.
- The automatic rebind value for APCOMPARE and APREUSE is NONE.
- If there is no existing value for the APPLCOMPAT bind option, the APPLCOMPAT subsystem parameter is used.
- If there is no existing value for the DESCSTAT bind option, the DESCSTAT subsystem parameter is used.

Automatic binds with package copies

If a package has previous or original copies as a result of rebinding with the PLANMGMT(BASIC) or PLANMGMT(EXTENDED) options or having the PLANMGMT subsystem parameter set to BASIC or EXTENDED, those copies are not affected by automatic rebind. Automatic rebind replaces only the current copy.

When automatic binds fail

If EXPLAIN(YES) was specified for the previous rebind operation, the ABEXP subsystem parameter controls whether Db2 captures EXPLAIN information during automatic rebinds. For details, see EXPLAIN PROCESSING field (ABEXP subsystem parameter) (Db2 Installation and Migration). Automatic rebinds fail for most EXPLAIN errors.

If an automatic bind occurs while running in ACCESS(MAINT) mode the automatic bind is run under the authorization id of SYSOPR. If SYSOPR is not defined as an installation SYSOPR the automatic bind fails.

Related concepts

Automatic binds in coexistence (Db2 Installation and Migration)

Application and SQL release incompatibilities (Db2 for z/OS What's New?)

Related tasks

Rebind old plans and packages in Db2 10 to avoid disruptive autobinds in Db2 11 (Db2 Installation and Migration)

Related reference

BIND PACKAGE (DSN) (Db2 Commands)

BIND PLAN (DSN) (Db2 Commands)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Specifying the rules that apply to SQL behavior at run time

You can specify whether Db2 rules or SQL standard rules apply to SQL behavior at run time.

About this task

Not only does SQLRULES specify the rules under which a type 2 CONNECT statement executes, but it also sets the initial value of the special register CURRENT RULES when the database server is the local Db2 system. When the server is not the local Db2 system, the initial value of CURRENT RULES is DB2. After binding a plan, you can change the value in CURRENT RULES in an application program by using the statement SET CURRENT RULES.

CURRENT RULES determines the SQL rules, Db2 or SQL standard, that apply to SQL behavior at run time. For example, the value in CURRENT RULES affects the behavior of defining check constraints by issuing the ALTER TABLE statement on a populated table:

- **If CURRENT RULES has a value of STD** and no existing rows in the table violate the check constraint, Db2 adds the constraint to the table definition. Otherwise, an error occurs and Db2 does not add the check constraint to the table definition.

  If the table contains data and is already in a check pending status, the ALTER TABLE statement fails.
- **If CURRENT RULES has a value of DB2**, Db2 adds the constraint to the table definition, defers the enforcing of the check constraints, and places the table space or partition in CHECK-pending status.

You can use the statement SET CURRENT RULES to control the action that the statement ALTER TABLE takes. Assuming that the value of CURRENT RULES is initially STD, the following SQL statements change the SQL rules to DB2, add a check constraint, defer validation of that constraint, place the table in CHECK-pending status, and restore the rules to STD.

```
EXEC SQL
  SET CURRENT RULES = 'DB2';
EXEC SQL
```

```
  ALTER TABLE DSN8B10.EMP
    ADD CONSTRAINT C1 CHECK (BONUS <= 1000.0);
EXEC SQL
  SET CURRENT RULES = 'STD';
```

See “Check constraints” on page 129 for information about check constraints.

You can also use CURRENT RULES in host variable assignments. For example, if you want to store the value of the CURRENT RULES special register at a particular point in time, you can use assign the value to a host variable, as in the following statement:

```
SET :XRULE = CURRENT RULES;
```

You can also use CURRENT RULES as the argument of a search-condition. For example, the following statement retrieves rows where the COL1 column contains the same value as the CURRENT RULES special register.

```
SELECT * FROM SAMPTBL WHERE COL1 = CURRENT RULES;
```

Db2 program preparation overview

Before you can run an application program on Db2 for z/OS, you need to prepare it. To prepare the program, create a load module, possibly one or more packages, and an application plan.

If your application program includes SQL statements, you need to process those SQL statements by using either the Db2 precompiler or the Db2 coprocessor that is provided with a compiler. Both the precompiler and the coprocessor perform the following actions:

- Replaces the SQL statements in your source programs with calls to Db2 language interface modules
- Creates a database request module (DBRM), which communicates your SQL requests to Db2 during the bind process

The following figure illustrates the program preparation process when you use the Db2 precompiler. After you process SQL statements in your source program by using the Db2 precompiler, you create a load module, possibly one or more packages, and an application plan. Creating a load module involves compiling the modified source code that is produced by the precompiler into an object program, and link-editing the object program to create a load module. Creating a package or an application plan, a process unique to Db2, involves binding one or more DBRMs, which are created by the Db2 precompiler, using the BIND PACKAGE command.

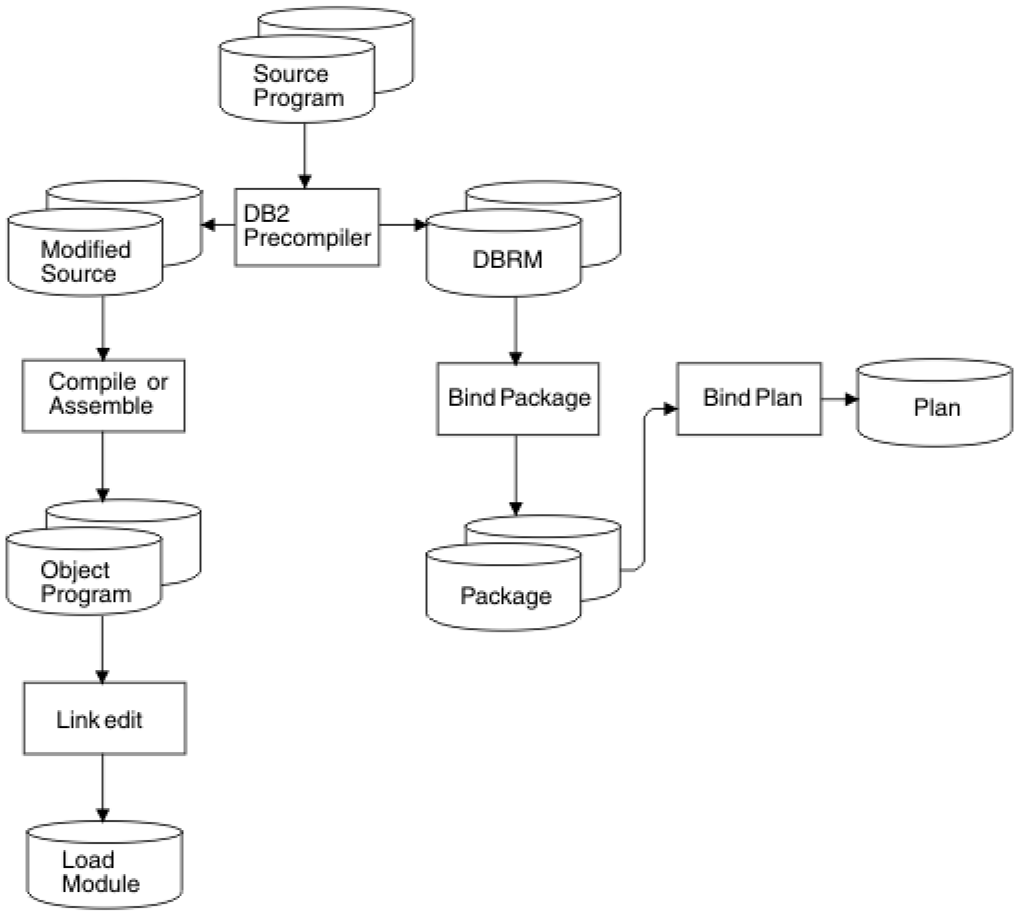


Figure 41. Program preparation with the Db2 precompiler

The following figure illustrates the program preparation process when you use the Db2 coprocessor. The process is similar to the process for the Db2 precompiler, except that the Db2 coprocessor does not create modified source for your application program.

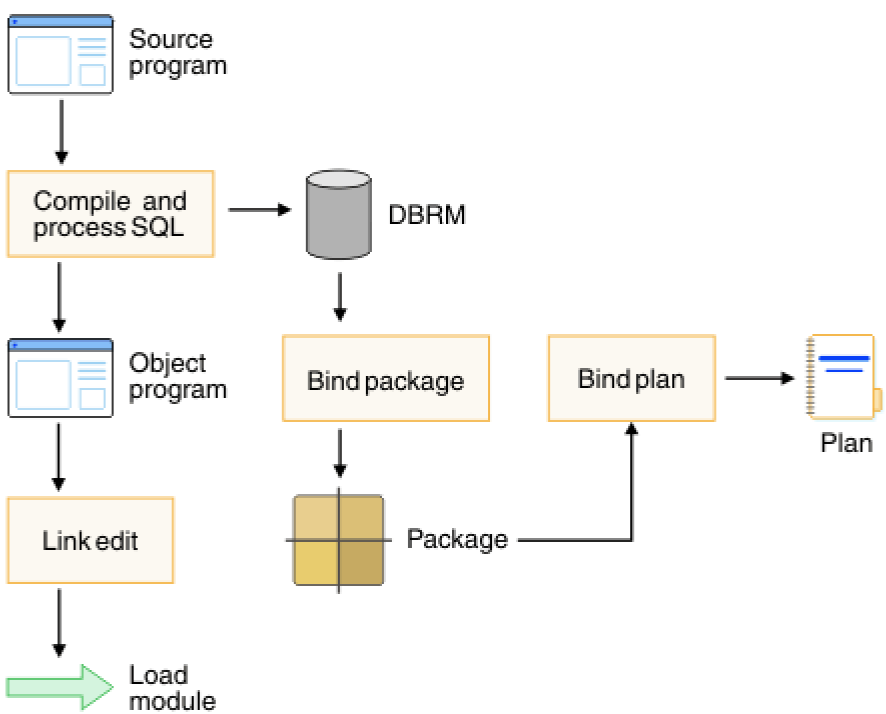


Figure 42. Program preparation with the Db2 coprocessor

Input and output data sets for DL/I batch jobs

DL/I batch jobs require an input data set with DD name DDITV02 and an output data set with DD name DDOTV02.

Db2 DL/I batch input:

Before you can run a DL/I batch job, you need to provide values for a number of input parameters. The input parameters are positional and delimited by commas.

You can specify values for the following parameters using a DDITV02 data set or a subsystem member:

```
SSN,LIT,ESMT,RTT,REO,CRC
```

You can specify values for the following parameters **only** in a DDITV02 data set:

```
CONNECTION_NAME,PLAN,PROG
```

If you use the DDITV02 data set and specify a subsystem member, the values in the DDITV02 DD statement override the values in the specified subsystem member. If you provide neither, Db2 abnormally terminates the application program with system abend code X'04E' and a unique reason code in register 15.

DDITV02 is the DD name for a data set that has DCB options of LRECL=80 and RECFM=F or FB.

A subsystem member is a member in the IMS procedure library. Its name is derived by concatenating the value of the SSM parameter to the value of the IMSID parameter. You specify the SSM parameter and the IMSID parameter when you invoke the DLIBATCH procedure, which starts the DL/I batch processing environment.

The meanings of the input parameters are:

SSN
: Specifies the name of the Db2 subsystem. This value is required. You must specify a name in order to make a connection to Db2.

  The SSN value can be from one to four characters long.

  If the value in the SSN parameter is the name of an active subsystem in the data sharing group, the application attaches to that subsystem. If the SSN parameter value is not the name of an active subsystem, but the value is a group attachment name, the application attaches to an active Db2 subsystem in the data sharing group.

LIT
: Specifies a language interface token. Db2 requires a language interface token to route SQL statements when operating in the online IMS environment. Because a batch application program can connect to only one Db2 system, Db2 does not use the LIT value.

  The LIT value can be from zero to four characters long.

  Recommendation: Specify the LIT value as SYS1.

  You can omit the LIT value by entering `SSN,,ESMT`.

ESMT
: Specifies the name of the Db2 initialization module, DSNMIN10. This value is required.

  The ESMT value must be eight characters long.

RTT
: Specifies the resource translation table. This value is optional.

  The RTT can be from zero to eight characters long.

REO
: Specifies the region error option. This option determines what to do if Db2 is not operational or the plan is not available. The three options are:

- *R*, the default, results in returning an SQL return code to the application program. The most common SQLCODE issued in this case is -923 (SQLSTATE '57015').
- *Q* results in an abend in the batch environment; however, in the online environment, this value places the input message in the queue again.
- *A* results in an abend in both the batch environment and the online environment.

If the application program uses the XRST call, and if coordinated recovery is required on the XRST call, REO is ignored. In that case, the application program terminates abnormally if Db2 is not operational.

The REO value can be from zero to one character long.

CRC
Specifies the command recognition character. Because Db2 commands are not supported in the DL/I batch environment, the command recognition character is not used at this time.

The CRC value can be from zero to one character long.

CONNECTION_NAME
Represents the name of the job step that coordinates Db2 activities. This value is optional. If you do not specify this option, the connection name defaults are:

Type of application
Default connection name

Batch job
Job name

Started task
Started task name

TSO user
TSO authorization ID

If a batch update job fails, you must use a separate job to restart the batch job. The connection name used in the restart job must be the same as the name that is used in the batch job that failed. Alternatively, if the default connection name is used, the restart job must have the same job name as the batch update job that failed.

Db2 requires unique connection names. If two applications try to connect with the same connection name, the second application program fails to connect to Db2.

The CONNECTION_NAME value can be from one to eight characters long.

PLAN
Specifies the Db2 plan name. This value is optional. If you do not specify the plan name, the application program module name is checked against the optional resource translation table. If the resource translation table has a match, the translated name is used as the Db2 plan name. If no match exists in the resource translation table, the application program module name is used as the plan name.

The PLAN value can be from zero to eight characters long.

PROG
Specifies the application program name. This value is required. It identifies the application program that is to be loaded and to receive control.

The PROG value can be from one to eight characters long.

Example: An example of the fields in the record is shown below:

```
DSN,SYS1,DSNMIN10,,R,-,BATCH001,DB2PLAN,PROGA
```

Db2 DL/I batch output:

In an online IMS environment, Db2 sends unsolicited status messages to the master terminal operator (MTO) and records on indoubt processing and diagnostic information to the IMS log. In a batch environment, Db2 sends this information to the output data set that is specified in the DDOTV02 DD statement. Ensure that the output data set has DCB options of RECFM=V or VB, LRECL=4092, and BLKSIZE of at least LRECL + 4. If the DD statement is missing, Db2 issues the message IEC130I and continues processing without any output.

You might want to save and print the data set, as the information is useful for diagnostic purposes. You can use the IMS module, DFSERA10, to print the variable-length data set records in both hexadecimal and character format.

Related concepts

Submitting work to be processed (Db2 Data Sharing Planning and Administration)

Db2-supplied JCL procedures for preparing an application

You can precompile and prepare an application program using a Db2-supplied JCL procedure.

Db2 has a unique JCL procedure for each supported language, with appropriate defaults for starting the Db2 precompiler and host language compiler or assembler. The procedures are in *prefix*.SDSNSAMP member DSNTIJMV, which installs the procedures.

Table 147. Procedures for precompiling programs

Language	Procedure	Invocation included in...
High-level assembler	DSNHASM	DSNTEJ2A
C	DSNHC	DSNTEJ2D
C++	DSNHCPPDSNHCPP2[2]	DSNTEJ2EN/A
Enterprise COBOL	DSNHICOB	DSNTEJ2C[1]
Fortran	DSNHFOR	DSNTEJ2F
PL/I	DSNHPLI	DSNTEJ2P
SQL	DSNHSQL	DSNTEJ63

Notes:

1. You must customize these programs to invoke the procedures that are listed in this table.
2. This procedure demonstrates how you can prepare an object-oriented program that consists of two data sets or members, both of which contain SQL.

If you use the PL/I macro processor, you must not use the PL/I *PROCESS statement in the source to pass options to the PL/I compiler. You can specify the needed options on the PARM.PLI= parameter of the EXEC statement in the DSNHPLI procedure.

JCL to include the appropriate interface code when using the Db2-supplied JCL procedures

To include the proper interface code when you submit the JCL procedures, use an INCLUDE SYSLIB statement in your link-edit JCL. The statement should specify the correct language interface module for the environment.

TSO, batch

```
//LKED.SYSIN  DD  *
  INCLUDE SYSLIB(member)
/*
```

member must be DSNELI or DSNULI, except for FORTRAN, in which case *member* must be DSNHFT.

IMS

```
//LKED.SYSIN  DD  *
  INCLUDE SYSLIB(DFSLI000)
  ENTRY (specification)
/*
```

DFSLI000 is the module for DL/I batch attach.

ENTRY *specification* varies depending on the host language. Include one of the following:

DLITCBL, for COBOL applications
PLICALLA, for PL/I applications
The program name, for assembler language applications.

Recommendation: For COBOL applications, specify the PSB linkage directly on the PROCEDURE DIVISION statement instead of on a DLITCBL entry point. When you specify the PSB linkage directly on the PROCEDURE DIVISION statement, you can either omit the ENTRY specification or specify the application program name instead of the DLITCBL entry point.

CICS

```
//LKED.SYSIN  DD  *
  INCLUDE SYSLIB(member)
/*
```

member must be DSNCLI or DSNULI.

Related concepts

"Universal language interface (DSNULI)" on page 117
The universal language interface (DSNULI) subcomponent determines the runtime environment and dynamically loads and branches to the appropriate language interface module.

Related tasks

Making the CAF language interface (DSNALI) available
Before you can invoke the call attachment facility (CAF), you must first make DSNALI available.

Compiling and link-editing an application
If you use the Db2 precompiler, your next step in the program preparation process is to compile and link-edit your program. As with the precompile step, you have a choice of methods.

Tailoring Db2-supplied JCL procedures for preparing CICS programs

Instead of using the Db2 Program Preparation panels to prepare your CICS program, you can tailor CICS-supplied JCL procedures to do that. To tailor a CICS procedure, you need to add some steps and change some DD statements.

About this task

Make changes as needed to perform the following actions:

- Process the program with the Db2 precompiler.
- Bind the application plan. You can do this any time after you precompile the program. You can bind the program either online by the DB2I panels or as a batch step in this or another z/OS job.
- Include a DD statement in the linkage editor step to access the Db2 load library.
- Be sure the linkage editor control statements contain an INCLUDE statement for the Db2 language interface module.

The following example illustrates the necessary changes. This example assumes the use of a COBOL program. For any other programming language, change the CICS procedure name and the Db2 precompiler options.

```
        //TESTC01 JOB
        //*
        //*********************************************************
        //*       DB2 PRECOMPILE THE COBOL PROGRAM
        //*********************************************************
(1)     //PC      EXEC PGM=DSNHPC,
(1)     //       PARM='HOST(COB2),XREF,SOURCE,FLAG(I),APOST'
(1)     //STEPLIB  DD  DISP=SHR,DSN=prefix.SDSNEXIT
(1)     //         DD  DISP=SHR,DSN=prefix.SDSNLOAD
(1)     //DBRMLIB  DD  DISP=OLD,DSN=USER.DBRMLIB.DATA(TESTC01)
(1)     //SYSCIN   DD  DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
(1)     //             SPACE=(800,(500,500))
(1)     //SYSLIB   DD  DISP=SHR,DSN=USER.SRCLIB.DATA
(1)     //SYSPRINT DD  SYSOUT=*
(1)     //SYSTERM  DD  SYSOUT=*
(1)     //SYSUDUMP DD  SYSOUT=*
(1)     //SYSUT1   DD  SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
(1)     //SYSUT2   DD  SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
(1)     //SYSIN    DD  DISP=SHR,DSN=USER.SRCLIB.DATA(TESTC01)
(1)     //*
```

```
        //********************************************************************
        //***    BIND THIS PROGRAM.
        //********************************************************************
(2)     //BIND    EXEC PGM=IKJEFT01,
(2)     //   COND=((4,LT,PC))
(2)     //STEPLIB  DD  DISP=SHR,DSN=prefix.SDSNEXIT
(2)     //         DD  DISP=SHR,DSN=prefix.SDSNLOAD
(2)     //DBRMLIB  DD  DISP=OLD,DSN=USER.DBRMLIB.DATA(TESTC01)
(2)     //SYSPRINT DD  SYSOUT=*
(2)     //SYSTSPRT DD  SYSOUT=*
(2)     //SYSUDUMP DD  SYSOUT=*
(2)     //SYSTSIN DD *
(2)      DSN S(DSN)
(2)      BIND PLAN(TESTC01)  MEMBER(TESTC01) ACTION(REP) RETAIN ISOLATION(CS)
(2)      END
        //********************************************************************
        //*            COMPILE THE COBOL PROGRAM
        //********************************************************************
(3)     //CICS    EXEC DFHEITVL
(4)     //TRN.SYSIN    DD  DSN=&&DSNHOUT,DISP=(OLD,DELETE)
(5)     //LKED.SYSLMOD DD  DSN=USER.RUNLIB.LOAD
(6)     //LKED.CICSLOAD DD  DISP=SHR,DSN=prefix.SDFHLOAD
        //LKED.SYSIN   DD  *
(7)      INCLUDE CICSLOAD(DSNCLI)
         NAME TESTC01(R)
        //********************************************************************
```

The procedure accounts for these steps:

Step 1. Precompile the program. The output of the Db2 precompiler becomes the input to the CICS command language translator.

Step 2. Bind the application plan.

Step 3. Call the CICS procedure to translate, compile, and link-edit a COBOL program. This procedure has several options that you need to consider.

Step 4. Reflect an application load library in the data set name of the SYSLMOD DD statement. You must include the name of this load library in the DFHRPL DD statement of the CICS run time JCL.

Step 5. Name the CICS load library that contains the module DSNCLI.

Step 6. Direct the linkage editor to include the CICS-Db2 language interface module (DSNCLI). In this example, the order of the various control sections (CSECTs) is of no concern because the structure of the procedure automatically satisfies any order requirements.

For more information about the procedure DFHEITVL, other CICS procedures, or CICS requirements for application programs, please see the appropriate CICS manual.

If you are preparing a particularly large or complex application, you can use another preparation method. For example, if your program requires four of your own link-edit include libraries, you cannot prepare the program with DB2I, because DB2I limits the number of include libraries to three, plus language, IMS or CICS, and Db2 libraries. Therefore, you would need another preparation method. **Be careful to use the correct language interface.**

Related reference

Data sets that the precompiler uses
When you invoke the precompiler you need to provide data sets that contain input for the precompiler, such as the host programming statements and SQL statements. You also need to provide data sets where the precompiler can store its output, such as the modified source code and diagnostics messages.

DB2I panels that are used for program preparation

DB2I contains a set of panels that let you prepare an application for execution.

The following table describes each of the panels that you need to use to prepare an application.

Table 148. DB2I panels used for program preparation

Panel name	Panel description
"Db2 Program Preparation panel" on page 889	Lets you choose specific program preparation functions to perform. For the functions that you choose, you can also display the associated panels to specify options for performing those functions. This panel also lets you change the DB2I default values and perform other precompile and prelink functions.
"DB2I Defaults Panel 1" on page 893	Lets you change many of the system defaults that are set at Db2 installation time.
"DB2I Defaults Panel 2" on page 895	Lets you change your default job statement and set additional COBOL options.
"Precompile panel" on page 896	Lets you specify values for precompile functions. You can reach this panel directly from the DB2I Primary Option Menu or from the Db2 Program Preparation panel. If you reach this panel from the Program Preparation panel, many of the fields contain values from the Primary and Precompile panels.
"Bind Package panel" on page 898	Lets you change many options when you bind a package. You can reach this panel directly from the DB2I Primary Option Menu or from the Db2 Program Preparation panel. If you reach this panel from the Db2 Program Preparation panel, many of the fields contain values from the Primary and Precompile panels.
"Bind Plan panel" on page 900	Lets you change options when you bind an application plan. You can reach this panel directly from the DB2I Primary Option Menu or as a part of the program preparation process. This panel also follows the Bind Package panels.
"Defaults for Bind Package and Defaults for Rebind Package panels" on page 903	Let you change the defaults for BIND or REBIND PACKAGE or PLAN.
"System Connection Types panel" on page 907	Lets you specify a system connection type. This panel displays if you choose to enable or disable connections on the Bind or Rebind Package or Plan panels.

Table 148. DB2I panels used for program preparation (continued)

Panel name	Panel description
"Panels for entering lists of values" on page 909	Let you enter or modify an unlimited number of values. A list panel looks similar to an ISPF edit session and lets you scroll and use a limited set of commands.
"Program Preparation: Compile, Link, and Run panel" on page 910	Lets you perform the last two steps in the program preparation process (compile and link-edit). This panel also lets you do the PL/I MACRO PHASE for programs that require this option. For TSO programs, the panel also lets you run programs.

Related reference

The DB2I primary option menu (Introduction to Db2 for z/OS)

DB2I panels that are used to rebind and free plans and packages
A set of DB2I panels lets you bind, rebind, or free packages.

Db2 Program Preparation panel

The Db2 Program Preparation panel lets you choose which specific program preparation function to perform.

For the functions you choose, you can also choose to display the associated panels to specify options for performing those functions. Some of the functions you can select are:

Precompile
The panel for this function lets you control the Db2 precompiler.

Bind a package
The panel for this function lets you bind your program's DBRM to a package and change your defaults for binding the packages.

Bind a plan
The panel for this function lets you create your program's application plan and change your defaults for binding the plans.

Compile, link, and run
The panel for these functions let you control the compiler or assembler and the linkage editor.

TSO and batch: For TSO programs, you can use the program preparation programs to control the host language run time processor and the program itself.

The Program Preparation panel also lets you change the DB2I default values, and perform other precompile and prelink functions.

On the Db2 Program Preparation panel, shown in the following figure, enter the name of the source program data set (this example uses SAMPLEPG.COBOL) and specify the other options you want to include. When finished, press ENTER to view the next panel.

```
DSNEPP01               DB2 PROGRAM PREPARATION               SSID: DSN
COMMAND ===>_

Enter the following:
 1  INPUT DATA SET NAME .... ===> SAMPLEPG.COBOL
 2  DATA SET NAME QUALIFIER  ===> TEMP       (For building data set names)
 3  PREPARATION ENVIRONMENT  ===> FOREGROUND (FOREGROUND, BACKGROUND, EDITJCL)
 4  RUN TIME ENVIRONMENT ... ===> TSO        (TSO, CAF, CICS, IMS, RRSAF)
 5  OTHER DSNH OPTIONS ..... ===>
                                            (Optional DSNH keywords)
Select functions:           Display panel?       Perform function?
 6  CHANGE DEFAULTS ........ ===> Y  (Y/N)
 7  PL/I MACRO PHASE ....... ===> N  (Y/N)         ===> N  (Y/N)
 8  PRECOMPILE ............. ===> Y  (Y/N)         ===> Y  (Y/N)
 9  CICS COMMAND TRANSLATION                       ===> N  (Y/N)
10  BIND PACKAGE ........... ===> Y  (Y/N)         ===> Y  (Y/N)
11  BIND PLAN............... ===> Y  (Y/N)         ===> Y  (Y/N)
12  COMPILE OR ASSEMBLE .... ===> Y  (Y/N)         ===> Y  (Y/N)
13  PRELINK................. ===> N  (Y/N)         ===> N  (Y/N)
14  LINK.................... ===> N  (Y/N)         ===> Y  (Y/N)
15  RUN..................... ===> N  (Y/N)         ===> Y  (Y/N)
```

Figure 43. The Db2 Program Preparation panel

The following explains the functions on the Db2 Program Preparation panel and how to complete the necessary fields in order to start program preparation.

1 INPUT DATA SET NAME
: Lets you specify the input data set name. The input data set name can be a PDS or a sequential data set, and can also include a member name. If you do not enclose the data set name in apostrophes, a standard TSO prefix (user ID) qualifies the data set name.

  The input data set name you specify is used to precompile, bind, link-edit, and run the program.

2 DATA SET NAME QUALIFIER
: Lets you qualify temporary data set names involved in the program preparation process. Use any character string from 1 to 8 characters that conforms to normal TSO naming conventions. (The default is TEMP.)

  For programs that you prepare in the background or that use EDITJCL for the PREPARATION ENVIRONMENT option, Db2 creates a data set named *tsoprefix.qualifier*.CNTL to contain the program preparation JCL. The name *tsoprefix* represents the prefix TSO assigns, and *qualifier* represents the value you enter in the DATA SET NAME QUALIFIER field. If a data set with this name already exists, Db2 deletes it.

3 PREPARATION ENVIRONMENT
: Lets you specify whether program preparation occurs in the foreground or background. You can also specify EDITJCL, in which case you are able to edit and then submit the job. Use:

  FOREGROUND to use the values you specify on the Program Preparation panel and to run immediately.

  BACKGROUND to create and submit a file containing a DSNH CLIST that runs immediately using the JOB control statement from either the DB2I Defaults panel or your site's SUBMIT exit. The file is saved.

  EDITJCL to create and open a file containing a DSNH CLIST in edit mode. You can then submit the CLIST or save it.

4 RUN TIME ENVIRONMENT
: Lets you specify the environment (TSO, CAF, CICS, IMS, RRSAF) in which your program runs.

  All programs are prepared under TSO, but can run in any of the environments. If you specify CICS, IMS, or RRSAF, then you must set the RUN field to NO because you cannot run such programs from the Program Preparation panel. If you set the RUN field to YES, you can specify only TSO or CAF.

  (Batch programs also run under the TSO Terminal Monitor Program. You therefore need to specify TSO in this field for batch programs.)

5 OTHER DSNH OPTIONS

Lets you specify a list of DSNH options that affect the program preparation process, and that override options specified on other panels. If you are using CICS, these can include options you want to specify to the CICS command translator.

If you specify options in this field, separate them by commas. You can continue listing options on the next line, but the total length of the option list can be no more than 70 bytes.

Fields 6 through 15 let you select the function to perform and to choose whether to show the DB2I panels for the functions you select. Use Y for YES, or N for NO.

If you are willing to accept default values for all the steps, enter N under Display panel? for all the other preparation panels listed.

To make changes to the default values, entering Y under Display panel? for any panel you want to see. DB2I then displays each of the panels that you request. After all the panels display, Db2 proceeds with the steps involved in preparing your program to run.

Variables for all functions used during program preparation are maintained separately from variables entered from the DB2I Primary Option Menu. For example, the bind plan variables you enter on the Program Preparation panel are saved separately from those on any Bind Plan panel that you reach from the Primary Option Menu.

6 CHANGE DEFAULTS

Lets you specify whether to change the DB2I defaults. Enter Y in the Display panel? field next to this option; otherwise enter N. Minimally, you should specify your subsystem identifier and programming language on the Defaults panel.

7 PL/I MACRO PHASE

Lets you specify whether to display the "Program Preparation: Compile, Link, and Run" panel to control the PL/I macro phase by entering PL/I options in the OPTIONS field of that panel. That panel also displays for options COMPILE OR ASSEMBLE, LINK, and RUN.

This field applies to PL/I programs only. If your program is not a PL/I program or does not use the PL/I macro processor, specify N in the Perform function field for this option, which sets the Display panel? field to the default N.

8 PRECOMPILE

Lets you specify whether to display the Precompile panel. To see this panel enter Y in the Display panel? field next to this option; otherwise enter N.

9 CICS COMMAND TRANSLATION

Lets you specify whether to use the CICS command translator. This field applies to CICS programs only.

IMS and TSO: If you run under TSO or IMS, ignore this step; this allows the Perform function field to default to N.

CICS: If you are using CICS and have precompiled your program, you must translate your program using the CICS command translator.

The command translator does not have a separate DB2I panel. You can specify translation options on the Other Options field of the Db2 Program Preparation panel, or in your source program if it is not an assembler program.

Because you specified a CICS run time environment, the Perform function column defaults to Y. Command translation takes place automatically after you precompile the program.

10 BIND PACKAGE

Lets you specify whether to display the Bind Package panel. To see it, enter Y in the Display panel? field next to this option; otherwise, enter N.

11 BIND PLAN

Lets you specify whether to display the Bind Plan panel. To see it, enter Y in the Display panel? field next to this option; otherwise, enter N.

12 COMPILE OR ASSEMBLE
Lets you specify whether to display the "Program Preparation: Compile, Link, and Run" panel. To see this panel enter Y in the Display panel? field next to this option; otherwise, enter N.

13 PRELINK
Lets you use the prelink utility to make your C, C++, or Enterprise COBOL for z/OS program reentrant. This utility concatenates compile-time initialization information from one or more text decks into a single initialization unit. To use the utility, enter Y in the Display panel? field next to this option; otherwise, enter N. If you request this step, then you must also request the compiler step and the link-edit step.

14 LINK
Lets you specify whether to display the "Program Preparation: Compile, Link, and Run" panel. To see it, enter Y in the Display panel? field next to this option; otherwise, enter N. If you specify Y in the Display panel? field for the COMPILE OR ASSEMBLE option, you do not need to make any changes to this field; the panel displayed for COMPILE OR ASSEMBLE is the same as the panel displayed for LINK. You can make the changes you want to affect the link-edit step at the same time you make the changes to the compiler step.

15 RUN
Lets you specify whether to run your program. The RUN option is available only if you specify TSO or CAF for RUN TIME ENVIRONMENT.

If you specify Y in the Display panel? field for the COMPILE OR ASSEMBLE or LINK option, you can specify N in this field, because the panel displayed for COMPILE OR ASSEMBLE and for LINK is the same as the panel displayed for RUN.

IMS and CICS: IMS and CICS programs cannot run using DB2I. If you are using IMS or CICS, use N in these fields.

TSO and batch: If you are using TSO and want to run your program, you must enter Y in the Perform function column next to this option. You can also indicate that you want to specify options and values to affect the running of your program, by entering Y in the Display panel column.

Pressing ENTER takes you to the first panel in the series you specified, in this example to the DB2I Defaults panel. If, at any point in your progress from panel to panel, you press the END key, you return to this first panel, from which you can change your processing specifications. Asterisks (*) in the Display panel? column of rows 7 through 14 indicate which panels you have already examined. You can see a panel again by writing a Y over an asterisk.

Related reference

Bind Package panel
The Bind Package panel is the first of two DB2I panels that request information about how you want to bind a package.

Bind Plan panel
The Bind Plan panel is the first of two DB2I panels that request information about how you want to bind an application plan.

DB2I Defaults Panel 1
DB2I Defaults Panel 1 lets you change many of the system default values that were set at Db2 installation time.

Defaults for Bind Package and Defaults for Rebind Package panels
These DB2I panels lets you change your defaults for BIND PACKAGE and REBIND PACKAGE options.

Defaults for Bind Plan and Defaults for Rebind Plan panels
These DB2I panels let you change your defaults for BIND PLAN and REBIND PLAN options.

Precompile panel
After you set the DB2I defaults, you can precompile your application. You can reach the Precompile panel by specifying it as a part of the program preparation process from the Db2 Program Preparation panel. Or you can reach it directly from the DB2I Primary Option Menu.

Program Preparation: Compile, Link, and Run panel

The Compile, Link, and Run panel lets you perform the last two steps in the program preparation process (compile and link-edit). This panel also lets you perform the PL/I MACRO PHASE for programs that require this option.

DSNH (TSO CLIST) (Db2 Commands)

Prelinking an application (z/OS Language Environment Programming Guide)

DB2I Defaults Panel 1

DB2I Defaults Panel 1 lets you change many of the system default values that were set at Db2 installation time.

The following figure shows the fields that affect the processing of the other DB2I panels.

```
DSNEOP01                DB2I DEFAULTS PANEL 1
COMMAND ===>_

Change defaults as desired:

 1  DB2 NAME ............. ===> DSN       (Subsystem identifier)
 2  DB2 CONNECTION RETRIES ===> 0         (How many retries for DB2 connection)
 3  APPLICATION LANGUAGE   ===> IBMCOB    (ASM, C, CPP, IBMCOB, FORTRAN, PLI)
 4  LINES/PAGE OF LISTING  ===> 60        (A number from 5 to 999)
 5  MESSAGE LEVEL ........ ===> I         (Information, Warning, Error, Severe)
 6  SQL STRING DELIMITER   ===> DEFAULT   (DEFAULT, ' or ")
 7  DECIMAL POINT ........ ===> .         (. or ,)
 8  STOP IF RETURN CODE >= ===> 8         (Lowest terminating return code)
 9  NUMBER OF ROWS         ===> 20        (For ISPF Tables)
 10  AS USER                ===>           (User ID to associate with trusted connection)
```

Figure 44. DB2I Defaults Panel 1

The following explains the fields on DB2I Defaults Panel 1.

1 Db2 NAME
Lets you specify the Db2 subsystem that processes your DB2I requests. If you specify a different Db2 subsystem, its identifier displays in the SSID (subsystem identifier) field located at the top, right side of your screen. The default is DSN.

2 Db2 CONNECTION RETRIES
Lets you specify the number of additional times to attempt to connect to Db2, if Db2 is not up when the program issues the DSN command. The program preparation process does not use this option.

Use a number from 0 to 120. The default is 0. Connections are attempted at 30-second intervals.

3 APPLICATION LANGUAGE
Lets you specify the default programming language for your application program. You can specify any of the following languages:

ASM
For High Level Assembler/z/OS

C
For C language

CPP
For C++

IBMCOB
For Enterprise COBOL for z/OS. This option is the default.

FORTRAN
For VS Fortran

PLI
For PL/I

If you specify IBMCOB, Db2 prompts you for more COBOL defaults on panel DSNEOP02. See "DB2I Defaults Panel 2" on page 895.

You cannot specify FORTRAN for IMS or CICS programs.

4 LINES/PAGE OF LISTING
Lets you specify the number of lines to print on each page of listing or SPUFI output. The default is 60.

5 MESSAGE LEVEL
Lets you specify the lowest level of message to return to you during the BIND phase of the preparation process. Use:

I
For all information, warning, error, and severe error messages

W
For warning, error, and severe error messages

E
For error and severe error messages

S
For severe error messages only

6 SQL STRING DELIMITER
Lets you specify the symbol used to delimit a string in SQL statements in COBOL programs. This option is valid only when the application language is IBMCOB. Use:

DEFAULT
To use the default defined at installation time

'
For an apostrophe

"
For a quotation mark

7 DECIMAL POINT
Lets you specify how your host language source program represents decimal separators and how SPUFI displays decimal separators in its output. Use a comma (,) or a period (.). The default is a period (.).

8 STOP IF RETURN CODE >=
Lets you specify the smallest value of the return code (from precompile, compile, link-edit, or bind) that will prevent later steps from running. Use:

4
To stop on warnings and more severe errors.

8
To stop on errors and more severe errors. The default is 8.

9 NUMBER OF ROWS
Lets you specify the default number of input entry rows to generate on the initial display of ISPF panels. The number of rows with non-blank entries determines the number of rows that appear on later displays.

10 AS USER
Lets you specify a user ID to associate with the trusted connection for the current DB2I session.

Db2 establishes the trusted connection for the user that you specify if the following conditions are true:

- The primary authorization ID that Db2 obtains after running the connection exit is allowed to use the trusted connection without authentication.
- The security label, if defined either implicitly or explicitly in the trusted context for the user, is defined in RACF for the user.

After Db2 establishes the trusted connection, the primary authorization ID, any secondary authorization IDs, any role, and any security label that is associated with the user ID that is specified in the AS USER field are used for the trusted connection. Db2 uses this security label to verify multilevel security for the user.

If the primary authorization ID that is associated with the user ID that is specified in the AS USER field is not allowed to use the trusted connection or requires authentication information, the connection request fails. If Db2 cannot verify the security label, the connection request also fails.

The value that you enter in this field is retained only for the length of the DB2I session. The field is reset to blank when you exit DB2I.

Suppose that the default programming language is PL/I and the default number of lines per page of program listing is 60. Your program is in COBOL, so you want to change field 3, APPLICATION LANGUAGE. You also want to print 80 lines to the page, so you need to change field 4, LINES/PAGE OF LISTING, as well. Figure 44 on page 893 shows the entries that you make in DB2I Defaults Panel 1 to make these changes. In this case, pressing ENTER takes you to Db2 Defaults Panel 2.

DB2I Defaults Panel 2

After you press Enter on the DB2I Defaults Panel 1, the DB2I Defaults Panel 2 is displayed. If you chose IBMCOB as the language on the DB2I Defaults Panel 1, three fields are displayed. Otherwise, only the first field is displayed.

The following figure shows the DB2I Defaults Panel 2 when IBMCOB is selected.

```
DSNEOP02                DB2I DEFAULTS PANEL 2
COMMAND ===>_

Change defaults as desired:

 1  DB2I JOB STATEMENT:   (Optional if your site has a SUBMIT exit)
    ===> //USRT001A JOB (ACCOUNT),'NAME'
    ===> //*
    ===> //*
    ===> //*

    COBOL DEFAULTS:                        (For IBMCOB)
 2  COBOL STRING DELIMITER ===> DEFAULT   (DEFAULT, ' or ")
 3  DBCS SYMBOL FOR DCLGEN ===> G         (G/N - Character in PIC clause)
```

Figure 45. DB2I Defaults Panel 2

1 DB2I JOB STATEMENT
Lets you change your default job statement. Specify a job control statement, and optionally, a JOBLIB statement to use either in the background or the EDITJCL program preparation environment. Use a JOBLIB statement to specify run time libraries that your application requires. If your program has a SUBMIT exit routine, Db2 uses that routine. If that routine builds a job control statement, you can leave this field blank.

2 COBOL STRING DELIMITER
Lets you specify the symbol used to delimit a string in a COBOL statement in a COBOL application. Use:

DEFAULT
To use the default defined at installation time

'
For an apostrophe

"
For a quotation mark

Leave this field blank to accept the default value.

3 DBCS SYMBOL FOR DCLGEN
Lets you enter either G (the default) or N, to specify whether DCLGEN generates a picture clause that has the form PIC G(*n*) DISPLAY-1 or PIC N(*n*).

Leave this field blank to accept the default value.

Pressing ENTER takes you to the next panel you specified on the Db2 Program Preparation panel, in this case, to the Precompile panel.

Precompile panel

After you set the DB2I defaults, you can precompile your application. You can reach the Precompile panel by specifying it as a part of the program preparation process from the Db2 Program Preparation panel. Or you can reach it directly from the DB2I Primary Option Menu.

The way you choose to reach the panel determines the default values of the fields it contains. The following figure shows the Precompile panel.

```
DSNETP01                     PRECOMPILE                       SSID: DSN
COMMAND ===>_

Enter precompiler data sets:
 1  INPUT DATA SET .... ===> SAMPLEPG.COBOL
 2  INCLUDE LIBRARY ... ===> SRCLIB.DATA

 3  DSNAME QUALIFIER .. ===> TEMP        (For building data set names)
 4  DBRM DATA SET ..... ===>

Enter processing options as desired:
 5  WHERE TO PRECOMPILE ===> FOREGROUND  (FOREGROUND, BACKGROUND, or EDITJCL)
 6  VERSION ........... ===>
                                         (Blank, VERSION, or AUTO)
 7  OTHER OPTIONS ..... ===>
```

Figure 46. The Precompile panel

The following explains the functions on the Precompile panel, and how to enter the fields for preparing to precompile.

1 INPUT DATA SET
Lets you specify the data set name of the source program and SQL statements to precompile.

If you reached this panel through the Db2 Program Preparation panel, this field contains the data set name specified there. You can override it on this panel.

If you reached this panel directly from the DB2I Primary Option Menu, you must enter the data set name of the program you want to precompile. The data set name can include a member name. If you do not enclose the data set name with apostrophes, a standard TSO prefix (user ID) qualifies the data set name.

2 INCLUDE LIBRARY
Lets you enter the name of a library containing members that the precompiler should include. These members can contain output from DCLGEN. If you do not enclose the name in apostrophes, a standard TSO prefix (user ID) qualifies the name.

You can request additional INCLUDE libraries by entering DSNH CLIST parameters of the form `P`*n*`LIB(`*dsname*`)`, where *n* is 2, 3, or 4) on the OTHER OPTIONS field of this panel or on the OTHER DSNH OPTIONS field of the Program Preparation panel.

3 DSNAME QUALIFIER
Lets you specify a character string that qualifies temporary data set names during precompile. Use any character string from 1 to 8 characters in length that conforms to normal TSO naming conventions.

If you reached this panel through the Db2 Program Preparation panel, this field contains the data set name qualifier specified there. You can override it on this panel.

If you reached this panel from the DB2I Primary Option Menu, you can either specify a DSNAME QUALIFIER or let the field take its default value, TEMP.

IMS and TSO: For IMS and TSO programs, Db2 stores the precompiled source statements (to pass to the compiler or assemble step) in a data set named *tsoprefix.qualifier.suffix*. A data set named *tsoprefix.qualifier*.PCLIST contains the precompiler print listing.

For programs prepared in the background or that use the PREPARATION ENVIRONMENT option EDITJCL (on the Db2 Program Preparation panel), a data set named *tsoprefix.qualifier*.CNTL contains the program preparation JCL.

In these examples, *tsoprefix* represents the prefix TSO assigns, often the same as the authorization ID. *qualifier* represents the value entered in the DSNAME QUALIFIER field. *suffix* represents the output name, which is one of the following: COBOL, FORTRAN, C, PLI, ASM, DECK, CICSIN, OBJ, or DATA. In the Precompile Panel that is shown above, the data set *tsoprefix*.TEMP.COBOL contains the precompiled source statements, and *tsoprefix*.TEMP.PCLIST contains the precompiler print listing. If data sets with these names already exist, then Db2 deletes them.

CICS: For CICS programs, the data set *tsoprefix.qualifier.suffix* receives the precompiled source statements in preparation for CICS command translation.

If you do not plan to do CICS command translation, the source statements in *tsoprefix.qualifier.suffix*, are ready to compile. The data set *tsoprefix.qualifier*.PCLIST contains the precompiler print listing.

When the precompiler completes its work, control passes to the CICS command translator. Because there is no panel for the translator, translation takes place automatically. The data set *tsoprefix.qualifier*.CXLIST contains the output from the command translator.

4 DBRM DATA SET

Lets you name the DBRM library data set for the precompiler output. The data set can also include a member name.

When you reach this panel, the field is blank. When you press ENTER, however, the value contained in the DSNAME QUALIFIER field of the panel, concatenated with *DBRM*, specifies the DBRM data set: *qualifier*.DBRM.

You can enter another data set name in this field only if you allocate and catalog the data set before doing so. This is true even if the data set name that you enter corresponds to what is otherwise the default value of this field.

The precompiler sends modified source code to the data set *qualifier.host*, where *host* is the language specified in the APPLICATION LANGUAGE field of DB2I Defaults panel 1.

5 WHERE TO PRECOMPILE

Lets you indicate whether to precompile in the foreground or background. You can also specify EDITJCL, in which case you are able to edit and then submit the job.

If you reached this panel from the Db2 Program Preparation panel, the field contains the preparation environment specified there. You can override that value if you want.

If you reached this panel directly from the DB2I Primary Option Menu, you can either specify a processing environment or allow this field to take its default value. Use:

FOREGROUND to immediately precompile the program with the values you specify in these panels.

BACKGROUND to create and immediately submit to run a file containing a DSNH CLIST using the JOB control statement from either DB2I Defaults Panel 2 or your site's SUBMIT exit. The file is saved.

EDITJCL to create and open a file containing a DSNH CLIST in edit mode. You can then submit the CLIST or save it.

6 VERSION

Lets you specify the version of the program and its DBRM. If the version contains the maximum number of characters permitted (64), you must enter each character with no intervening blanks from one line to the next. This field is optional.

7 OTHER OPTIONS
Lets you enter any option that the DSNH CLIST accepts, which gives you greater control over your program. The DSNH options you specify in this field override options specified on other panels. The option list can continue to the next line, but the total length of the list can be no more than 70 bytes.

Related reference
DSNH (TSO CLIST) (Db2 Commands)

Bind Package panel

The Bind Package panel is the first of two DB2I panels that request information about how you want to bind a package.

You can reach the Bind Package panel either directly from the DB2I Primary Option Menu, or as a part of the program preparation process. If you enter the Bind Package panel from the Program Preparation panel, many of the Bind Package entries contain values from the Primary and Precompile panels. Figure 47 on page 898 shows the Bind Package panel.

```
DSNEBP07                 BIND PACKAGE                       SSID: DSN

 COMMAND ===>_

 Specify output location and collection names:
  1  LOCATION NAME ............. ===>                 (Defaults to local)
  2  COLLECTION-ID ............. ===>               > (Required)
 Specify package source (DBRM or COPY):
  3   DBRM:        COPY:         ===> DBRM      (Specify DBRM or COPY)
  4  MEMBER   or  COLLECTION-ID  ===>         >
  5  PASSWORD or  PACKAGE-ID ..  ===>         >
  6  LIBRARY  or  VERSION .....  ===>
                                                (Blank, or COPY version-id)
  7  ........ --  OPTIONS .....  ===>           (COMPOSITE or COMMAND)
 Enter options as desired:
  8  CHANGE CURRENT DEFAULTS?    ===> NO        (NO or YES)
  9  ENABLE/DISABLE CONNECTIONS? ===> NO        (NO or YES)
 10  OWNER OF PACKAGE (AUTHID).. ===>         > (Leave blank for primary ID)
 11  QUALIFIER ................  ===>         > (Leave blank for OWNER)
 12  ACTION ON PACKAGE ........  ===> REPLACE   (ADD or REPLACE)
 13  INCLUDE PATH? ............  ===> NO        (NO or YES)
 14  REPLACE VERSION ..........  ===>           (Replacement version-id)
```

Figure 47. The Bind Package panel

The following information explains the functions on the Bind Package panel and how to fill the necessary fields in order to bind your program.

1 LOCATION NAME
Lets you specify the system at which to bind the package. You can use from 1 to 16 characters to specify the location name. The location name must be defined in the catalog table SYSIBM.LOCATIONS. The default is the local DBMS.

2 COLLECTION-ID
Lets you specify the collection the package is in. You can use from 1 to 128 characters to specify the collection, and the first character must be alphabetic. This field is scrollable.

3 DBRM: COPY:
Lets you specify whether you are creating a new package (DBRM) or making a copy of a package that already exists (COPY). Use:

DBRM
To create a new package. You must specify values in the LIBRARY, PASSWORD, and MEMBER fields.

COPY
To copy an existing package. You must specify values in the COLLECTION-ID and PACKAGE-ID fields. (The VERSION field is optional.)

4 MEMBER or COLLECTION-ID

MEMBER (for new packages): If you are creating a new package, this option lets you specify the DBRM to bind. You can specify a member name from 1 to 128 characters. This field is scrollable. The default name depends on the input data set name.

- If the input data set is partitioned, the default name is the member name of the input data set specified in the INPUT DATA SET NAME field of the Db2 Program Preparation panel.
- If the input data set is sequential, the default name is the second qualifier of this input data set.

COLLECTION-ID (for copying a package): If you are copying a package, this option specifies the collection ID that contains the original package. You can specify a collection ID from 1 to 128 characters, which must be different from the collection ID specified on the PACKAGE ID field. This field is scrollable.

5 PASSWORD or PACKAGE-ID

PASSWORD (for new packages): If you are creating a new package, this lets you enter password for the library you list in the LIBRARY field. You can use this field only if you reached the Bind Package panel directly from the Db2 Primary Option Menu. This field is scrollable.

PACKAGE-ID (for copying packages): If you are copying a package, this option lets you specify the name of the original package. You can enter a package ID from 1 to 128 characters. This field is scrollable.

6 LIBRARY or VERSION

LIBRARY (for new packages): If you are creating a new package, this lets you specify the names of the libraries that contain the DBRMs specified on the MEMBER field for the bind process. Libraries are searched in the order specified and must in the catalog tables.

VERSION (for copying packages): If you are copying a package, this option lets you specify the version of the original package. You can specify a version ID from 1 to 64 characters.

7 OPTIONS

Lets you specify which bind options Db2 uses when you issue BIND PACKAGE with the COPY option. Specify:

- **COMPOSITE (default)** to cause Db2 to use any options you specify in the BIND PACKAGE command. For all other options, Db2 uses the options of the copied package.
- **COMMAND** to cause Db2 to use the options you specify in the BIND PACKAGE command. For all other options, Db2 uses the following values:
  - For a local copy of a package, Db2 uses the defaults for the local Db2 subsystem.
  - For a remote copy of a package, Db2 uses the defaults for the server on which the package is bound.

8 CHANGE CURRENT DEFAULTS?

Lets you specify whether to change the current defaults for binding packages. If you enter YES in this field, you see the Defaults for Bind Package panel as your next step. You can enter your new preferences there; for instructions, see "Defaults for Bind Package and Defaults for Rebind Package panels" on page 903.

9 ENABLE/DISABLE CONNECTIONS?

Lets you specify whether you want to enable and disable system connections types to use with this package. This is valid only if the LOCATION NAME field names your local Db2 system.

Placing YES in this field displays a panel (shown in Figure 53 on page 908) that lets you specify whether various system connections are valid for this application. You can specify connection names to further identify enabled connections within a connection type. A connection name is valid only when you also specify its corresponding connection type.

The default enables all connection types.

10 OWNER OF PACKAGE (AUTHID)
Lets you specify the primary authorization ID of the owner of the new package. That ID is the name owning the package, and the name associated with all accounting and trace records produced by the package.

The owner must have the privileges required to run SQL statements contained in the package.

The default is the primary authorization ID of the bind process.

The field is scrollable, and the maximum field length is 128.

11 QUALIFIER
Lets you specify the default schema for unqualified tables, views, indexes, and aliases. You can specify a schema name from 1 to 128 characters. The default is the authorization ID of the package owner. This field is scrollable.

12 ACTION ON PACKAGE
Lets you specify whether to replace an existing package or create a new one. Use:

REPLACE (default) to replace the package named in the PACKAGE-ID field if it already exists, and add it if it does not. (Use this option if you are changing the package because the SQL statements in the program changed. If only the SQL environment changes but not the SQL statements, you can use REBIND PACKAGE.)

ADD to add the package named in the PACKAGE-ID field, only if it does not already exist.

13 INCLUDE PATH?
Indicates whether you will supply a list of schema names that Db2 searches when it resolves unqualified distinct type, user-defined function, and stored procedure names in SQL statements. The default is NO. If you specify YES, Db2 displays a panel in which you specify the names of schemas for Db2 to search.

14 REPLACE VERSION
Lets you specify whether to replace a specific version of an existing package or create a new one. If the package and the version named in the PACKAGE-ID and VERSION fields already exist, you must specify REPLACE. You can specify a version ID from 1 to 64 characters. The default version ID is that specified in the VERSION field.

Bind Plan panel

The Bind Plan panel is the first of two DB2I panels that request information about how you want to bind an application plan.

Like the Precompile panel, you can reach the Bind Plan panel either directly from the DB2I Primary Option Menu, or as a part of the program preparation process. You must have an application plan, even if you bind your application to packages; this panel also follows the Bind Package panels.

If you enter the Bind Plan panel from the Program Preparation panel, many of the Bind Plan entries contain values from the Primary and Precompile panels.

```
DSNEBP02                 BIND PLAN                          SSID: DSN
 COMMAND ===>_

Enter primary package list:
  1  LOCATION NAME ............. ===>            (Defaults to local)
  2  COLLECTION ID ............. ===>          > (Required)
  3  PACKAGE ID ................ ===>            (Package ID or *)
  4  ADDITIONAL PACKAGE LISTS .. ===> NO         (YES to include more packages)

 Enter options as desired:
  5  PLAN NAME ................. ===>            (Required to create a plan)
  6  CHANGE CURRENT DEFAULTS?..  ===> NO         (NO or YES)
  7  ENABLE/DISABLE CONNECTIONS? ===> NO         (NO or YES)
  8  OWNER OF PLAN (AUTHID)..... ===>          > (Leave blank for your primary ID)
  9  QUALIFIER ................. ===>          > (For tables, views, and aliases)
 10  CACHESIZE ................. ===> 0          (Blank, or value 0-4096)
 11  ACTION ON PLAN ............ ===> REPLACE    (REPLACE or ADD)
 12  RETAIN EXECUTION AUTHORITY. ===> NO         (YES to retain user list)
 13  CURRENT SERVER ............ ===>            (Location name)
 14  INCLUDE PATH? ............. ===> NO         (NO or YES)
```

Figure 48. The Bind Plan panel

The following explains the functions on the Bind Plan panel and how to fill the necessary fields in order to bind your program.

1 LOCATION NAME
Lets you specify the remote system where the package that is named in the PACKAGE ID field is bound. The location name must be defined in the catalog table SYSIBM.LOCATIONS. The default is the local DBMS.

2 COLLECTION ID
Lets you specify the collection that includes the package that is to be bound into the plan.

The field is scrollable, and the maximum field length is 128.

3 PACKAGE ID
Lets you specify the name of the package that is to be bound into the plan.

4 ADDITIONAL PACKAGE LISTS
Lets you include a list of additional packages in the plan. If you specify YES, a separate panel displays, where you must enter the package location, collection name, and package name for each package to include in the plan. This list is optional.

5 PLAN NAME
Lets you name the application plan to create. You can specify a name from 1 to 8 characters, and the first character must be alphabetic. If there are no errors, the bind process prepares the plan and enters its description into the EXPLAIN table.

If you reached this panel through the Db2 Program Preparation panel, the default for this field depends on the value you entered in the INPUT DATA SET NAME field of that panel.

If you reached this panel directly from the Db2 Primary Option Menu, you must include a plan name if you want to create an application plan. The default name for this field depends on the input data set:

- If the input data set is partitioned, the default name is the member name.
- If the input data set is sequential, the default name is the second qualifier of the data set name.

6 CHANGE CURRENT DEFAULTS?
Lets you specify whether to change the current defaults for binding plans. If you enter YES in this field, you see the Defaults for Bind Plan panel as your next step. You can enter your new preferences there.

7 ENABLE/DISABLE CONNECTIONS?
Lets you specify whether you want to enable and disable system connections types to use with this package. This is valid only if the LOCATION NAME field names your local Db2 system.

Placing YES in this field displays a panel (shown in Figure 53 on page 908) that lets you specify whether various system connections are valid for this application. You can specify connection names to further identify enabled connections within a connection type. A connection name is valid only when you also specify its corresponding connection type.

The default enables all connection types.

8 OWNER OF PLAN (AUTHID)
Lets you specify the primary authorization ID of the owner of the new plan. That ID is the name owning the plan, and the name associated with all accounting and trace records produced by the plan.

The owner must have the privileges required to run SQL statements contained in the plan.

The field is scrollable, and the maximum field length is 128.

9 QUALIFIER
Lets you specify the default schema for unqualified tables, views, and aliases. You can specify a schema name from 1 to 128 characters, which must conform to the rules for SQL identifiers. If you leave this field blank, the default qualifier is the authorization ID of the plan owner. This field is scrollable.

Lets you specify the default schema for unqualified tables, views, and aliases. You can specify a schema name from 1 to 8 characters, which must conform to the rules for SQL identifiers. If you leave this field blank, the default qualifier is the authorization ID of the plan owner.

10 CACHESIZE
Lets you specify the size (in bytes) of the authorization cache. Valid values are in the range 0 to 4096. Values that are not multiples of 256 round up to the next highest multiple of 256. A value of 0 indicates that Db2 does not use an authorization cache. The default is 1024.

Each concurrent user of a plan requires 8 bytes of storage, with an additional 32 bytes for overhead.

11 ACTION ON PLAN
Lets you specify whether this is a new or changed application plan. Use:

REPLACE (default) to replace the plan named in the PLAN NAME field if it already exists, and add the plan if it does not exist.
ADD to add the plan named in the PLAN NAME field, only if it does not already exist.

12 RETAIN EXECUTION AUTHORITY
Lets you choose whether or not those users with the authority to bind or run the existing plan are to keep that authority over the changed plan. This applies only when you are replacing an existing plan.

If the plan ownership changes and you specify YES, the new owner grants BIND and EXECUTE authority to the previous plan owner.

If the plan ownership changes and you do not specify YES, then everyone but the new plan owner loses EXECUTE authority (but not BIND authority), and the new plan owner grants BIND authority to the previous plan owner.

13 CURRENT SERVER
Lets you specify the initial server to receive and process SQL statements in this plan. You can specify a name from 1 to 16 characters, which you must previously define in the catalog table SYSIBM.LOCATIONS.

If you specify a remote server, Db2 connects to that server when the first SQL statement executes. The default is the name of the local Db2 subsystem.

14 INCLUDE PATH?
Indicates whether you will supply a list of schema names that Db2 searches when it resolves unqualified distinct type, user-defined function, and stored procedure names in SQL statements. The default is NO. If you specify YES, Db2 displays a panel in which you specify the names of schemas for Db2 to search.

When you finish making changes to this panel, press ENTER to go to the second of the program preparation panels, Program Prep: Compile, Link, and Run.

Related tasks

Caching authorization IDs for plans (Managing Security)

Related reference

Defaults for Bind Plan and Defaults for Rebind Plan panels
These DB2I panels let you change your defaults for BIND PLAN and REBIND PLAN options.

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Defaults for Bind Package and Defaults for Rebind Package panels

These DB2I panels lets you change your defaults for BIND PACKAGE and REBIND PACKAGE options.

On the following panel, enter new defaults for binding a package.

```
DSNEBP10                DEFAULTS FOR BIND PACKAGE             SSID: DSN
 COMMAND ===> _
  ----------------- Use the UP/DOWN keys to access all options ------------------
                                                                    More:     +
 Change default options as necessary:

  1  ISOLATION LEVEL ......... ===>            (CS, RR, RS, UR, or NC)
  2  VALIDATION TIME ......... ===>            (RUN or BIND)
  3  RESOURCE RELEASE TIME ... ===>            (COMMIT, DEALLOCATE, or
                                                  INHERITFROMPLAN)
  4  EXPLAIN PATH SELECTION .. ===>            (NO or YES)
  5  DATA CURRENCY ........... ===>            (NO or YES)
  6  PARALLEL DEGREE ......... ===>            (1 or ANY)
  7  SQLERROR PROCESSING ..... ===>            (NOPACKAGE or CONTINUE)
  8  REOPTIMIZE FOR INPUT VARS ===>            (ALWAYS, NONE, ONCE, or AUTO)
  9  DEFER PREPARE ........... ===>            (NO, YES, or INHERITFROMPLAN)
 10  KEEP DYN SQL PAST COMMIT  ===>            (NO or YES)
 11  APPLICATION ENCODING ...  ===>            (Blank, ASCII, EBCDIC,
                                                 UNICODE, or ccsid)
 12  OPTIMIZATION HINT ......  ===>          > (Blank or 'hint-id')
 13  IMMEDIATE WRITE ......... ===>            (YES, NO, or INHERITFROMPLAN)
 14  DYNAMIC RULES ........... ===>            (RUN, BIND, DEFINE, or INVOKE)
 15  DBPROTOCOL .............. ===>            (blank, DRDA, or DRDACBF)
 16  ACCESS PATH REUSE ....... ===> NONE       (ERROR, NONE, or WARN)
 17  ACCESS PATH COMPARISON .. ===> NONE       (ERROR, NONE, or WARN)
 18  SYSTEM_TIME SENSITIVE ... ===>            (blank, NO, or YES)
 19  BUSINESS_TIME SENSITIVE . ===>            (blank, NO, or YES)
 20  ARCHIVE SENSITIVE ....... ===>            (blank, NO, or YES)
 21  APPLICATION COMPATIBILITY ===>            (blank, V10R1, or V11R1)
--------------------------------------------------------------------------------
 PRESS:  ENTER to continue   UP/DOWN to scroll   RETURN to EXIT
```

Figure 49. The Defaults for Bind Package panel

On the following panel, enter new defaults for rebinding a package.

With a few minor exceptions, the options on this panel are the same as the options for the defaults for rebinding a package. However, the defaults for REBIND PACKAGE are different from those shown in the preceding figure, and you can specify SAME in any field to specify the values used the last time the package was bound. For rebinding, the default value for all fields is SAME.

```
DSNEBP11                DEFAULTS FOR REBIND PACKAGE           SSID: DSN
 COMMAND ===> _
  ----------------- Use the UP/DOWN keys to access all options ------------------
                                                                    More:     +
 Change default options as necessary:

  1  ISOLATION LEVEL ......... ===>            (SAME, CS, RR, RS, UR, or NC)
  2  PLAN VALIDATION TIME .... ===>            (SAME, RUN, or BIND)
  3  RESOURCE RELEASE TIME ... ===>            (SAME, DEALLOCATE, COMMIT,
                                                 or INHERITFROMPLAN)
  4  EXPLAIN PATH SELECTION .. ===>            (SAME, NO, or YES)
  5  DATA CURRENCY ........... ===>            (SAME, NO, or YES)
  6  PARALLEL DEGREE ......... ===>            (SAME, 1 or ANY)
  7  REOPTIMIZE FOR INPUT VARS ===>            (SAME, ALWAYS, NONE, ONCE, AUTO)
  8  DEFER PREPARE ........... ===>            (SAME, NO, YES,
                                                 or INHERITFROMPLAN)
  9  KEEP DYN SQL PAST COMMIT  ===>            (SAME, NO, or YES)
 10  APPLICATION ENCODING ...  ===>            (SAME, Blank, ASCII, EBCDIC,
                                                 UNICODE, or ccsid)
 11  OPTIMIZATION HINT ......  ===>          > (Blank or 'hint-id')
 12  IMMEDIATE WRITE ......... ===>            (SAME, NO, YES,
                                                 or INHERITFROMPLAN)
 13  DBPROTOCOL .............. ===>            (blank, DRDA, or DRDACBF)
 14  DYNAMIC RULES ........... ===>            (SAME, RUN, BIND,
                                                 DEFINERUN, DEFINEBIND,
                                                 INVOKERUN or INVOKEBIND)
 15  PLAN MANAGEMENT ......... ===>            (DEFAULT, BASIC, EXTENDED, OFF)
 16  ACCESS PATH REUSE ....... ===>            (DEFAULT, ERROR, NONE, or WARN)
 17  ACCESS PATH COMPARISON .. ===>            (DEFAULT, ERROR, NONE, or WARN)
 18  ACCESS PATH RETAIN DUPS . ===>            (DEFAULT, NO, OR YES)
 19  SYSTEM_TIME SENSITIVE ... ===>            (SAME, NO, or YES)
 20  BUSINESS_TIME SENSITIVE . ===>            (SAME, NO, or YES)
 21  ARCHIVE SENSITIVE ....... ===>            (SAME, NO, or YES)
 22  APPLICATION COMPATIBILITY ===>            (SAME, V10R1, or V11R1)
-------------------------------------------------------------------------------
 PRESS:  ENTER to continue   UP/DOWN to scroll   RETURN to EXIT
```

Figure 50. The Defaults for Rebind Package panel

The following table lists the fields on the Defaults for Bind Package and Defaults for Rebind Package panels, and the corresponding bind and rebind options.

Table 149. Defaults for Bind Package and Defaults for Rebind Package panel fields and corresponding bind or rebind options

Field name	**Bind or rebind option**
ACCESS PATH COMPARISON	APCOMPARE
ACCESS PATH RETAIN DUPS	APRETAINDUP
ACCESS PATH REUSE	APREUSE
APPLICATION COMPATIBILITY	APPLCOMPAT
APPLICATION ENCODING	ENCODING
ARCHIVE SENSITIVE	ARCHIVESENSITIVE
BUSINESS_TIME SENSITIVE	BUSTINESENSITIVE
DATA CURRENCY	CURRENTDATA
DBPROTOCOL	DBPROTOCOL
DEFER PREPARE	DEFER and NODEFER
DYNAMIC RULES	DYNAMICRULES
EXPLAIN PATH SELECTION	EXPLAIN
IMMEDIATE WRITE	IMMEDWRITE
ISOLATION LEVEL	ISOLATION

Table 149. Defaults for Bind Package and Defaults for Rebind Package panel fields and corresponding bind or rebind options (continued)

Field name	Bind or rebind option
KEEP DYN SQL PAST COMMIT	KEEPDYNAMIC
OPTIMIZATION HINT	OPTHINT
PARALLEL DEGREE	DEGREE
PLAN MANAGEMENT	PLANMGMT
REOPTIMIZE FOR INPUT VARS	REOPT
RESOURCE RELEASE TIME	RELEASE
SQLERROR PROCESSING	SQLERROR
SYSTEM_TIME SENSITIVE	SYSTIMESENSITIVE
VALIDATION TIME and PLAN VALIDATION TIME	VALIDATE

Related concepts

Dynamic rules options for dynamic SQL statements
The DYNAMICRULES bind option and the runtime environment determine the rules for the dynamic SQL attributes.

Parallel processing (Db2 Performance)

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Related tasks

Setting the isolation level of SQL statements in a REXX program
Isolation levels specify the locking behavior for SQL statements. You can set the isolation level for SQL statements in your REXX program to repeatable read (RR), read stability (RS), cursor stability (CS), or uncommitted read (UR).

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Defaults for Bind Plan and Defaults for Rebind Plan panels

These DB2I panels let you change your defaults for BIND PLAN and REBIND PLAN options.

On the following panel, enter new defaults for binding a plan.

```
DSNEBP10                DEFAULTS FOR BIND PLAN                SSID: DSN
 COMMAND ===>

 Change default options as necessary:

  1  ISOLATION LEVEL ......... ===> RR         (RR, RS, CS, or UR)
  2  VALIDATION TIME ......... ===> RUN        (RUN or BIND)
  3  RESOURCE RELEASE TIME ... ===> COMMIT     (COMMIT, DEALLOCATE, or
                                               INHERITFROMPLAN)
  4  EXPLAIN PATH SELECTION .. ===> NO         (NO or YES)
  5  DATA CURRENCY ........... ===> NO         (NO or YES)
  6  PARALLEL DEGREE ......... ===> 1          (1 or ANY)
  7  RESOURCE ACQUISITION TIME ===> USE        (USE or ALLOCATE)
  8  REOPTIMIZE FOR INPUT VARS ===> NONE       (ALWAYS, NONE, ONCE, AUTO)
  9  DEFER PREPARE ........... ===> NO         (NO, YES, INHERITFROMPLAN)
 10  KEEP DYN SQL PAST COMMIT. ===> NO         (NO or YES)
 11  APPLICATION ENCODING ...  ===>            (Blank,ASCII,EBCDIC,UNICODE,
                                                or ccsid)
 12  OPTIMIZATION HINT ......  ===>          > (Blank or 'hint-id')
 13  IMMEDIATE WRITE ......... ===> NO         (NO, YES, INHERITFROMPLAN)
 14  DYNAMIC RULES ........... ===> RUN        (RUN or BIND)
 15  SQLRULES................. ===> DB2        (DB2 or STD)
 16  DISCONNECT .............. ===> EXPLICIT   (EXPLICIT, AUTOMATIC,
                                                or CONDITIONAL)
 17 PROGRAM AUTHORIZATION ...  ===> DISABLE    (DISABLE, ENABLE)
```

Figure 51. The Defaults for Bind Plan panel

On the following panel, enter new defaults for rebinding a plan.

```
DSNEBP11                DEFAULTS FOR REBIND PLAN              SSID: DSN
 COMMAND ===>

 Change default options as necessary:

  1  ISOLATION LEVEL ......... ===> SAME       (SAME, RR, RS, CS, or UR)
  2  PLAN VALIDATION TIME .... ===> SAME       (SAME, RUN, or BIND)
  3  RESOURCE RELEASE TIME ... ===> SAME       (SAME, DEALLOCATE, COMMIT,
                                                 or INHERITFROMPLAN)
  4  EXPLAIN PATH SELECTION .. ===> SAME       (SAME, NO, or YES)
  5  DATA CURRENCY ........... ===> SAME       (SAME, NO, or YES)
  6  PARALLEL DEGREE ......... ===> SAME       (SAME, 1 or ANY)
  7  REOPTIMIZE FOR INPUT VARS ===> SAME       (SAME, ALWAYS, NONE, ONCE, AUTO)
  8  DEFER PREPARE ........... ===> SAME       (SAME, NO, YES,
                                                 or INHERITFROMPLAN)
  9  KEEP DYN SQL PAST COMMIT. ===> SAME       (SAME, NO, or YES)
 10  APPLICATION ENCODING ...  ===> SAME       (SAME,Blank,ASCII,EBCDIC,
                                                 UNICODE, or ccsid)
 11  OPTIMIZATION HINT ......  ===>          > (SAME, 'hint-id')
 12  IMMEDIATE WRITE ........  ===> SAME       (SAME, NO, YES,
                                                 or INHERITFROMPLAN)
 13  SQLRULES ...............  ===> SAME       (SAME, DB2 or STD)
 14  DYNAMIC RULES ........... ===> SAME       (SAME, RUN, or BIND)
 15  RESOURCE ACQUISITION TIME ===> SAME       (SAME, ALLOCATE, or USE)
 16  DISCONNECT .............. ===> SAME       (SAME, EXPLICIT, AUTOMATIC,
                                                or CONDITIONAL)
 17  PROGRAM AUTHORIZATION ... ===> SAME       (SAME, DISABLE, ENABLE)
```

Figure 52. The Defaults for Rebind Plan panel

The following table lists the fields on the Defaults for Bind Package and Defaults for Rebind Package, and the corresponding bind and rebind options.

Table 150. Defaults for Bind Plan and Defaults for Rebind Plan panel fields and corresponding bind or rebind options

Field name	Bind or rebind option
APPLICATION ENCODING	ENCODING
DATA CURRENCY	CURRENTDATA
DBPROTOCOL	DBPROTOCOL

Table 150. Defaults for Bind Plan and Defaults for Rebind Plan panel fields and corresponding bind or rebind options (continued)

Field name	Bind or rebind option
DEFER PREPARE	DEFER and NODEFER
DISCONNECT	DISCONNECT
DYNAMIC RULES	DYNAMICRULES
EXPLAIN PATH SELECTION	EXPLAIN
IMMEDIATE WRITE	IMMEDWRITE
ISOLATION LEVEL	ISOLATION
KEEP DYN SQL PAST COMMIT	KEEPDYNAMIC
OPTIMIZATION HINT	OPTHINT
PARALLEL DEGREE	DEGREE
PROGRAM AUTHORIZATION	PROGAUTH
REOPTIMIZE FOR INPUT VARS	REOPT
RESOURCE ACQUISITION TIME	ACQUIRE
RESOURCE RELEASE TIME	RELEASE
VALIDATION TIME and PLAN VALIDATION TIME	VALIDATE

Related concepts

Dynamic rules options for dynamic SQL statements
The DYNAMICRULES bind option and the runtime environment determine the rules for the dynamic SQL attributes.

Parallel processing (Db2 Performance)

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Related tasks

Caching authorization IDs for plans (Managing Security)

Setting the isolation level of SQL statements in a REXX program
Isolation levels specify the locking behavior for SQL statements. You can set the isolation level for SQL statements in your REXX program to repeatable read (RR), read stability (RS), cursor stability (CS), or uncommitted read (UR).

Specifying the rules that apply to SQL behavior at run time
You can specify whether Db2 rules or SQL standard rules apply to SQL behavior at run time.

System Connection Types panel

The System Connection Types panel lets you specify which types of connections can use a plan or package.

This panel displays if you enter YES for ENABLE/DISABLE CONNECTIONS? on the Bind or Rebind Package or Plan panels. For the Bind or Rebind Package panel, the REMOTE option does not display as it does in the following panel.

```
DSNEBP13    SYSTEM CONNECTION TYPES FOR BIND ...              SSID: DSN
 COMMAND ===>

 Select system connection types to be Enabled/Disabled:

  1  ENABLE ALL CONNECTION TYPES? ===>    (* to enable all types)
 or
  2  ENABLE/DISABLE SPECIFIC CONNECTION TYPES ===>    (E/D)

     BATCH ....... ===>    (Y/N)          SPECIFY CONNECTION NAMES?
     DB2CALL ..... ===>    (Y/N)
     RRSAF ....... ===>    (Y/N)
     CICS ........ ===>    (Y/N)          ===> N  (Y/N)
     IMS ......... ===>    (Y/N)
     DLIBATCH .... ===>    (Y/N)          ===> N  (Y/N)
     IMSBMP ...... ===>    (Y/N)          ===> N  (Y/N)
     IMSMPP ...... ===>    (Y/N)          ===> N  (Y/N)
     REMOTE ...... ===>    (Y/N)
```

Figure 53. The System Connection Types panel

To enable or disable connection types (that is, allow or prevent the connection from running the package or plan), enter the following information.

1 ENABLE ALL CONNECTION TYPES?
Lets you enter an asterisk (*) to enable all connections. After that entry, you can ignore the rest of the panel.

2 ENABLE/DISABLE SPECIFIC CONNECTION TYPES
Lets you specify a list of types to enable or disable; you cannot enable some types and disable others in the same operation. If you list types to enable, enter E; that disables all other connection types. If you list types to disable, enter D; that enables all other connection types.

For each connection type that follows, enter Y (yes) if it is on your list, N (no) if it is not. The connection types are:

- **BATCH** for a TSO connection
- **DB2CALL** for a CAF connection
- **RRSAF** for an RRSAF connection
- **CICS** for a CICS connection
- **IMS** for all IMS connections: DLIBATCH, IMSBMP, and IMSMPP
- **DLIBATCH** for a DL/I Batch Support Facility connection
- **IMSBMP** for an IMS connection to a BMP region
- **IMSMPP** for an IMS connection to an MPP or IFP region
- **REMOTE** for all remote locations or no remote locations

For each connection type that has a second arrow, under SPECIFY CONNECTION NAMES?, enter Y if you want to list specific connection names of that type. Leave N (the default) if you do not. If you use Y in any of those fields, you see another panel on which you can enter the connection names.

If you use the DISPLAY command under TSO on this panel, you can determine what you have currently defined as "enabled" or "disabled" in your ISPF DSNSPFT library (member DSNCONNS). The information does not reflect the current state of the Db2 Catalog.

If you type DISPLAY ENABLED on the command line, you get the connection names that are currently enabled for your TSO connection types. For example:

```
Display OF ALL   connection name(s) to be  ENABLED

       CONNECTION           SUBSYSTEM
       CICS1                ENABLED
       CICS2                ENABLED
       CICS3                ENABLED
       CICS4                ENABLED
       DLI1                 ENABLED
       DLI2                 ENABLED
       DLI3                 ENABLED
```

```
        DLI4                    ENABLED
        DLI5                    ENABLED
```

Related reference

Panels for entering lists of values
Some fields in DB2I panels are associated with command keywords that accept multiple values. Those fields lead you to a list panel that lets you enter or modify multiple values.

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Panels for entering lists of values

Some fields in DB2I panels are associated with command keywords that accept multiple values. Those fields lead you to a list panel that lets you enter or modify multiple values.

A list panel looks like an ISPF edit session and lets you scroll and use a limited set of commands.

The format of each list panel varies, depending on the content and purpose for the panel. The following figure shows a generic sample of a list panel:

```
panelid         Specific subcommand function                 SSID: DSN
 COMMAND ===>_                                 SCROLL ===>

 Subcommand operand values:

  CMD
 """"     value  …
 """"     value  …
 """"
 """"
 """"
 """"
```

Figure 54. Generic example of a DB2I list panel

All of the list panels let you enter limited commands in two places:

- On the system command line, prefixed by ====>
- In a special command area, identified by """"

On the system command line, you can use:

END
: Saves all entered variables, exits the table, and continues to process.

CANCEL
: Discards all entered variables, terminates processing, and returns to the previous panel.

SAVE
: Saves all entered variables and remains in the table.

In the special command area, you can use:

I*nn*
: Insert *nn* lines after this one.

D*nn*
: Delete this and the following lines for *nn* lines.

R*nn*
: Repeat this line *nn* number of times.

The default for *nn* is 1.

When you finish with a list panel, specify END to same the current panel values and continue processing.

Program Preparation: Compile, Link, and Run panel

The Compile, Link, and Run panel lets you perform the last two steps in the program preparation process (compile and link-edit). This panel also lets you perform the PL/I MACRO PHASE for programs that require this option.

For TSO programs, the panel also lets you run programs.

```
DSNEPP02    PROGRAM PREP: COMPILE, PRELINK, LINK, AND RUN     SSID: DSN
 COMMAND ===>_

 Enter compiler or assembler options:
  1  INCLUDE LIBRARY ===> SRCLIB.DATA
  2  INCLUDE LIBRARY ===>
  3  OPTIONS ....... ===> NUM, OPTIMIZE, ADV

 Enter linkage editor options:
  4  INCLUDE LIBRARY ===> SAMPLIB.COBOL
  5  INCLUDE LIBRARY ===>
  6  INCLUDE LIBRARY ===>
  7  LOAD LIBRARY .. ===> RUNLIB.LOAD
  8  PRELINK OPTIONS ===>
  9  LINK OPTIONS... ===>
 Enter run options:
 10  PARAMETERS .... ===> D01, D02, D03/
 11  SYSIN DATA SET  ===> TERM
 12  SYSPRINT DS ... ===> TERM
```

Figure 55. The Program Preparation: Compile, Link, and Run panel

1,2 INCLUDE LIBRARY
Lets you specify up to two libraries containing members for the compiler to include. The members can also be output from DCLGEN. You can leave these fields blank. There is no default.

3 OPTIONS
Lets you specify compiler, assembler, or PL/I macro processor options. You can also enter a list of compiler or assembler options by separating entries with commas, blanks, or both. You can leave these fields blank. There is no default.

4,5,6 INCLUDE LIBRARY
Lets you enter the names of up to three libraries containing members for the linkage editor to include. You can leave these fields blank. There is no default.

7 LOAD LIBRARY
Lets you specify the name of the library to hold the load module. The default value is RUNLIB.LOAD.

If the load library specified is a PDS, and the input data set is a PDS, the member name specified in INPUT DATA SET NAME field of the Program Preparation panel is the load module name. If the input data set is sequential, the second qualifier of the input data set is the load module name.

You must complete this field if you request LINK or RUN on the Program Preparation panel.

8 PRELINK OPTIONS
Lets you enter a list of prelinker options. Separate items in the list with commas, blanks, or both. You can leave this field blank. There is no default.

The prelink utility applies only to programs using C, C++, and Enterprise COBOL for z/OS.

9 LINK OPTIONS
Lets you enter a list of link-edit options. Separate items in the list with commas, blanks, or both.

To prepare a program that uses 31-bit addressing and runs above the 16-megabyte line, specify the following link-edit options: AMODE=31, RMODE=ANY.

10 PARAMETERS
Lets you specify a list of parameters you want to pass either to your host language run time processor, or to your application. Separate items in the list with commas, blanks, or both. You can leave this field blank.

If you are preparing an IMS or CICS program, you must leave this field blank; you cannot use DB2I to run IMS and CICS programs.

Use a slash (/) to separate the options for your run time processor from those for your program.

- For PL/I and Fortran, run time processor parameters must appear on the left of the slash, and the application parameters must appear on the right.

```
run time processor parameters / application parameters
```

- For COBOL, reverse this order. run time processor parameters must appear on the right of the slash, and the application parameters must appear on the left.
- For assembler and C, there is no supported run time environment, and you need not use a slash to pass parameters to the application program.

11 SYSIN DATA SET
Lets you specify the name of a SYSIN (or in Fortran, FT05F001) data set for your application program, if it needs one. If you do not enclose the data set name in apostrophes, a standard TSO prefix (user ID) and suffix is added to it. The default for this field is TERM.

If you are preparing an IMS or CICS program, you must leave this field blank; you cannot use DB2I to run IMS and CICS programs.

12 SYSPRINT DS
Lets you specify the names of a SYSPRINT (or in Fortran, FT06F001) data set for your application program, if it needs one. If you do not enclose the data set name in apostrophes, a standard TSO prefix (user ID) and suffix is added to it. The default for this field is TERM.

If you are preparing an IMS or CICS program, you must leave this field blank; you cannot use DB2I to run IMS and CICS programs.

Your application could need other data sets besides SYSIN and SYSPRINT. If so, remember to catalog and allocate them before you run your program.

When you press ENTER after entering values in this panel, Db2 compiles and link-edits the application. If you specified in the Db2 Program Preparation panel that you want to run the application, Db2 also runs the application.

Related reference

Prelinking an application (z/OS Language Environment Programming Guide)

DB2I panels that are used to rebind and free plans and packages

A set of DB2I panels lets you bind, rebind, or free packages.

The following table describes additional panels that you can use to rebind and free packages and plans. It also describes the Run panel, which you can use to run application programs that have already been prepared.

Table 151. DB2I panels used to rebind and free plans and packages and used to Run application programs

Panel	Panel description
“Bind/Rebind/Free Selection panel” on page 912	The BIND/REBIND/FREE panel lets you select the BIND, REBIND, or FREE, PLAN, PACKAGE, or TRIGGER PACKAGE process that you need.
“Rebind Package panel” on page 913	The Rebind Package panel lets you change options when you rebind a package.
“Rebind Trigger Package panel” on page 915	The Rebind Trigger Package panel lets you change options when you rebind a trigger package.
“Rebind Plan panel” on page 916	The Rebind Plan panel lets you change options when you rebind an application plan.

Table 151. DB2I panels used to rebind and free plans and packages and used to Run application programs (continued)

Panel	Panel description
"Free Package panel" on page 918	The Free Package panel lets you change options when you free a package.
"Free Plan panel" on page 919	The Free Plan panel lets you change options when you free an application plan.
"DB2I Run panel" on page 922	The Run panel lets you start an application program. You should use this panel if you have already prepared the program and you only want to run it. You can also run a program by using the "Program Prep: Compile, Prelink, Link, and Run" panel.

Related reference

DB2I panels that are used for program preparation
DB2I contains a set of panels that let you prepare an application for execution.

The DB2I primary option menu (Introduction to Db2 for z/OS)

Bind/Rebind/Free Selection panel

The Bind/Rebind/Free selection panel lets choose whether to bind, rebind, or free plans and packages.

```
DSNEBP01                  BIND/REBIND/FREE                    SSID: DSN
 COMMAND ===>_

 Select one of the following and press ENTER:

  1  BIND PLAN                (Add or replace an application plan)

  2  REBIND PLAN              (Rebind existing application plan or plans)

  3  FREE PLAN                (Erase application plan or plans)

  4  BIND PACKAGE             (Add or replace a package)

  5  REBIND PACKAGE           (Rebind existing package or packages)

  6  REBIND TRIGGER PACKAGE   (Rebind existing package or packages)

  7  FREE PACKAGE     (Erase a package or packages)
```

Figure 56. The Bind/Rebind/Free selection panel

This panel lets you select the process you need.

1 BIND PLAN
: Lets you build an application plan. You must have an application plan to allocate Db2 resources and support SQL requests during run time. If you select this option, the Bind Plan panel displays. For more information, see "Bind Plan panel" on page 900.

2 REBIND PLAN
: Lets you rebuild an application plan when changes to it affect the plan but the SQL statements in the program are the same. For example, you should rebind when you change authorizations, create a new index that the plan uses, or use RUNSTATS. If you select this option, the Rebind Plan panel displays. For more information, see "Rebind Plan panel" on page 916.

3 FREE PLAN
: Lets you delete plans from Db2. If you select this option, the Free Plan panel displays. For more information, see "Free Plan panel" on page 919.

4 BIND PACKAGE
Lets you build a package. If you select this option, the Bind Package panel displays. For more information, see "Bind Package panel" on page 898.

5 REBIND PACKAGE
Lets you rebuild a package when changes to it affect the package but the SQL statements in the program are the same. For example, you should rebind when you change authorizations, create a new index that the package uses, or use RUNSTATS. If you select this option, the Rebind Package panel displays. For more information, see "Rebind Package panel" on page 913.

6 REBIND TRIGGER PACKAGE
Lets you rebuild a trigger package when you need to change options for the package. When you execute CREATE TRIGGER, Db2 binds a trigger package using a set of default options. You can use REBIND TRIGGER PACKAGE to change those options. For example, you can use REBIND TRIGGER PACKAGE to change the isolation level for the trigger package. If you select this option, the Rebind Trigger Package panel displays. For more information, see "Rebind Trigger Package panel" on page 915.

7 FREE PACKAGE
Lets you delete a specific version of a package, all versions of a package, or whole collections of packages from Db2. If you select this option, the Free Package panel displays. For more information, see "Free Package panel" on page 918.

Rebind Package panel

The Rebind Package panel is the first of two panels that you use to rebind a package. This panel lets you specify options for rebinding the package.

The following figure shows the rebind package options.

```
DSNEBP08                  REBIND PACKAGE                      SSID: DSN
 COMMAND ===>_

  1  Rebind all local packages   ===>           (* to rebind all packages)

 or
     Enter package name(s) to be rebound:
  2  LOCATION NAME ............. ===>                  (Defaults to local)
  3  COLLECTION-ID ............. ===>                > (Required)
  4  PACKAGE-ID ................ ===>          >         (Required)
  5  VERSION-ID ................ ===>
                                                (*, Blank, (), or version-id)
  6  ADDITIONAL PACKAGES? ...... ===>           (Yes to include more packages)

 Enter options as desired ...... ===>
  7  CHANGE CURRENT DEFAULTS?... ===>           (NO or YES)
  8  OWNER OF PACKAGE (AUTHID).. ===>         > (SAME, new OWNER)
  9  QUALIFIER ................. ===>         > (SAME, new QUALIFIER)
 10  ENABLE/DISABLE CONNECTIONS? ===>           (NO or YES)
 11  INCLUDE PATH? ............. ===>           (SAME, DEFAULT, or YES)
```

Figure 57. The Rebind Package panel

This panel lets you choose options for rebinding a package.

1 Rebind all local packages
Lets you rebind all packages on the local DBMS. To do so, place an asterisk (*) in this field; otherwise, leave it blank.

2 LOCATION NAME
Lets you specify where to bind the package. If you specify a location name, you should use from 1 to 16 characters, and you must have defined it in the catalog table SYSIBM.LOCATIONS.

3 COLLECTION-ID
Lets you specify the collection of the package to rebind. You must specify a collection ID from 1 to 128 characters, or an asterisk (*) to rebind all collections in the local Db2 system. You cannot use the asterisk to rebind a remote collection. This field is scrollable.

4 PACKAGE-ID
Lets you specify the name of the package to rebind. You must specify a package ID from 1 to 8 characters, or an asterisk (*) to rebind all packages in the specified collections in the local Db2 system. You cannot use the asterisk to rebind a remote package.

The field is scrollable, and the maximum field length is 128.

5 VERSION-ID
Lets you specify the version of the package to rebind. You must specify a version ID from 1 to 64 characters, or an asterisk (*) to rebind all versions in the specified collections and packages in the local Db2 system. You cannot use the asterisk to rebind a remote version.

6 ADDITIONAL PACKAGES?
Lets you indicate whether to name more packages to rebind. Use YES to specify more packages on an additional panel, described on "Panels for entering lists of values" on page 909. The default is NO.

7 CHANGE CURRENT DEFAULTS?
Lets you indicate whether to change the binding defaults. Use:

NO (default) to retain the binding defaults of the previous package.

YES to change the binding defaults from the previous package. For information about the defaults for binding packages, see "Defaults for Bind Package and Defaults for Rebind Package panels" on page 903.

8 OWNER OF PACKAGE (AUTHID)
Lets you change the authorization ID for the package owner. The owner must have the required privileges to execute the SQL statements in the package. The default is the existing package owner.

The field is scrollable, and the maximum field length is 128.

9 QUALIFIER
Lets you specify the default schema for all unqualified table names, views, indexes, and aliases in the package. You can specify a schema name from 1 to 8 characters, which must conform to the rules for the SQL short identifier. The default is the existing qualifier name.

The field is scrollable, and the maximum field length is 128.

10 ENABLE/DISABLE CONNECTIONS?
Lets you specify whether you want to enable and disable system connections types to use with this package. This is valid only if the LOCATION NAME field names your local Db2 system.

Placing YES in this field displays a panel (shown in Figure 53 on page 908) that lets you specify whether various system connections are valid for this application.

The default is the values used for the previous package.

11 INCLUDE PATH?
Indicates which one of the following actions you want to perform:

- Request that Db2 uses the same schema names as when the package was bound for resolving unqualified distinct type, user-defined function, and stored procedure names in SQL statements. Choose SAME to perform this action. This is the default.
- Supply a list of schema names that Db2 searches when it resolves unqualified distinct type, user-defined function, and stored procedure names in SQL statements. Choose YES to perform this action.
- Request that Db2 resets the SQL path to SYSIBM, SYSFUN, SYSPROC, and the package owner. Choose DEFAULT to perform this action.

If you specify YES, Db2 displays a panel in which you specify the names of schemas for Db2 to search.

Related reference
BIND and REBIND options for packages, plans, and services (Db2 Commands)

Rebind Trigger Package panel

The Rebind Trigger Package panel specifies options for rebinding a trigger package.

The following figure shows those options.

```
DSNEBP19                REBIND TRIGGER PACKAGE                SSID: DSN
 COMMAND ===>_

  1  Rebind all trigger packages ===>           (* to rebind all packages)

 or
     Enter trigger package name(s) to be rebound:
  2  LOCATION NAME ............. ===>                  (Defaults to local)
  3  COLLECTION-ID (SCHEMA NAME) ===>               >  (Required)
  4  PACKAGE-ID (TRIGGER NAME).. ===>               >  (Required)

 Enter options as desired ...... ===>
  5  ISOLATION LEVEL ........... ===> SAME       (SAME, RR, RS, CS, UR, or NC)
  6  RESOURCE RELEASE TIME ..... ===> SAME       (SAME, DEALLOCATE, or COMMIT)
  7  EXPLAIN PATH SELECTION .... ===> SAME       (SAME, NO, or YES)
  8  DATA CURRENCY ............. ===> SAME       (SAME, NO, or YES)
  9  IMMEDIATE WRITE OPTION .... ===> SAME       (SAME, NO, YES)
  10 PLAN MANAGEMENT ........... ===> DEFAULT    (DEFAULT, BASIC, EXTENDED, OFF)
  11 ACCESS PATH REUSE ......... ===> DEFAULT    (DEFAULT, ERROR, NONE, or WARN)
  12 ACCESS PATH COMPARISON .... ===> DEFAULT    (DEFAULT, ERROR, NONE, or WARN)
  13 ACCESS PATH RETAIN DUPS ... ===> DEFAULT    (DEFAULT, NO, or YES)
  14 SYSTEM_TIME SENSITIVE ..... ===> SAME    (SAME, NO, or YES)
  15 BUSINESS_TIME SENSITIVE ... ===> SAME    (SAME, NO, or YES)
  16 ARCHIVE SENSITIVE ......... ===> SAME    (SAME, NO, or YES)
  17 APPLICATION COMPATIBILITY . ===> SAME       (SAME, V10R1, V11R1)
```

Figure 58. The Rebind Trigger Package panel

This panel lets you choose options for rebinding a trigger package.

1 Rebind all trigger packages
Lets you rebind all packages on the local DBMS. To do so, place an asterisk (*) in this field; otherwise, leave it blank.

2 LOCATION NAME
Lets you specify where to bind the trigger package. If you specify a location name, you should use from 1 to 16 characters, and you must have defined it in the catalog table SYSIBM.LOCATIONS.

3 COLLECTION-ID (SCHEMA NAME)
Lets you specify the collection of the trigger package to rebind. You must specify a collection ID from 1 to 128 characters, or an asterisk (*) to rebind all collections in the local Db2 system. You cannot use the asterisk to rebind a remote collection. This field is scrollable.

4 PACKAGE-ID
Lets you specify the name of the trigger package to rebind. You must specify a package ID from 1 to 128 characters, or an asterisk (*) to rebind all trigger packages in the specified collections in the local Db2 system. You cannot use the asterisk to rebind a remote trigger package. This field is scrollable.

5 ISOLATION LEVEL
Lets you specify how far to isolate your application from the effects of other running applications. The default is the value used for the old trigger package.

6 RESOURCE RELEASE TIME
Lets you specify COMMIT or DEALLOCATE to tell when to release locks on resources. The default is that used for the old trigger package.

7 EXPLAIN PATH SELECTION
Lets you specify YES or NO for whether to obtain EXPLAIN information about how SQL statements in the package execute. The default is the value used for the old trigger package.

The bind process inserts information into the table *owner*.PLAN_TABLE, where *owner* is the authorization ID of the plan or package owner. If you defined *owner*.DSN_STATEMNT_TABLE, Db2 also inserts information about the cost of statement execution into that table. If you specify YES in this field and BIND in the VALIDATION TIME field, and if you do not correctly define PLAN_TABLE, the bind fails.

8 DATA CURRENCY
Lets you specify YES or NO for whether you need data currency for ambiguous cursors opened at remote locations. The default is the value used for the old trigger package.

Data is current if the data within the host structure is identical to the data within the base table. Data is always current for local processing.

9 IMMEDIATE WRITE OPTION
Specifies when Db2 writes the changes for updated group buffer pool-dependent pages. This field applies only to a data sharing environment. The values that you can specify are:

SAME
Choose the value of IMMEDIATE WRITE that you specified when you bound the trigger package. SAME is the default.

NO
Write the changes at or before phase 1 of the commit process. If the transaction is rolled back later, write the additional changes that are caused by the rollback at the end of the abort process.

PH1 is equivalent to NO.

YES
Write the changes immediately after group buffer pool-dependent pages are updated.

10 PLAN MANAGEMENT
Specifies the PLANMGMT option to use for rebinding the trigger. DEFAULT means to take the default setting for this option when rebinding for the old trigger package.

11 ACCESS PATH REUSE
Specifies the APREUSE option to use for rebinding the trigger. DEFAULT means to take the default setting for this option when rebinding for the old trigger package.

12 ACCESS PATH COMPARISON
Specifies the APCOMPARE option to use for rebinding the trigger. DEFAULT means to take the default setting for this option when rebinding for the old trigger package.

13 ACCESS PATH RETAIN DUPS
Specifies the APRETAINDUP option to use for rebinding the trigger. DEFAULT means to take the default setting for this option when rebinding for the old trigger package.

14 SYSTEM_TIME SENSITIVE
Specifies the SYSTIMESENSITIVE option to use for rebinding the trigger. SAME means to take the previous setting for this option when rebinding the old trigger package.

15 BUSINESS_TIME SENSITIVE
Specifies the BUSTIMESENSITIVE option to use for rebinding the trigger. SAME means to take the previous setting for this option when rebinding the old trigger package.

16 ARCHIVE SENSITIVE
Specifies the ARCHIVESENSITIVE option to use for rebinding the trigger. SAME means to take the previous setting for this option when rebinding the old trigger package.

17 APPLICATION COMPATIBILITY
Specifies the APPLCOMPAT option to use for rebinding the trigger. SAME means to take the previous setting for this option when rebinding the old trigger package.

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Rebind Plan panel

The Rebind Plan panel is the first of two panels that you use to rebind a plan. This panel lets you specify options for rebinding the plan.

The following figure shows the rebind plan options.

```
DSNEBP03                  REBIND PLAN                          SSID: DSN
 COMMAND ===>_

 Enter plan name(s) to be rebound:
  1  PLAN NAME ................. ===>           (* to rebind all plans)
  2  ADDITIONAL PLANS? ......... ===> NO        (Yes to include more plans)

 Enter options as desired:
  3  CHANGE CURRENT DEFAULTS?... ===> NO        (NO or YES)
  4  OWNER OF PLAN (AUTHID)..... ===> SAME    > (SAME, new OWNER)
  5  QUALIFIER ................. ===> SAME    > (SAME, new QUALIFIER)
  6  CACHESIZE ................. ===> SAME      (SAME, or value 0-4096)
  7  ENABLE/DISABLE CONNECTIONS? ===> NO        (NO or YES)
  8  INCLUDE PACKAGE LIST?...... ===> SAME      (SAME, NO, or YES)
  9  CURRENT SERVER ............ ===>                     (Location name)
 10  INCLUDE PATH? ............. ===> SAME      (SAME, DEFAULT, or YES)
```

Figure 59. The Rebind Plan panel

This panel lets you specify options for rebinding your plan.

1 PLAN NAME
: Lets you name the application plan to rebind. You can specify a name from 1 to 8 characters, and the first character must be alphabetic. Do not begin the name with DSN, because it could create name conflicts with Db2. If there are no errors, the bind process prepares the plan and enters its description into the EXPLAIN table.

  If you leave this field blank, the bind process occurs but produces no plan.

2 ADDITIONAL PLANS?
: Lets you indicate whether to name more plans to rebind. Use YES to specify more plans on an additional panel, described at "Panels for entering lists of values" on page 909. The default is NO.

3 CHANGE CURRENT DEFAULTS?
: Lets you indicate whether to change the binding defaults. Use:

  NO (default) to retain the binding defaults of the previous plan.

  YES to change the binding defaults from the previous plan.

4 OWNER OF PLAN (AUTHID)
: Lets you change the authorization ID for the plan owner. The owner must have the required privileges to execute the SQL statements in the plan. The default is the existing plan owner.

  The field is scrollable, and the maximum field length is 128.

5 QUALIFIER
: Lets you specify the default schema for all unqualified table names, views, indexes, and aliases in the plan. You can specify a schema name from 1 to 128 characters, which must conform to the rules for the SQL identifier. The default is the authorization ID. This field is scrollable.

6 CACHESIZE
: Lets you specify the size (in bytes) of the authorization cache. Valid values are in the range 0 to 4096. Values that are not multiples of 256 round up to the next highest multiple of 256. A value of 0 indicates that Db2 does not use an authorization cache. The default is the cache size specified for the previous plan.

  Each concurrent user of a plan requires 8 bytes of storage, with an additional 32 bytes for overhead.

7 ENABLE/DISABLE CONNECTIONS?
: Lets you specify whether you want to enable and disable system connections types to use with this plan. This is valid only for rebinding on your local Db2 system.

  Placing YES in this field displays a panel (shown in Figure 53 on page 908) that lets you specify whether various system connections are valid for this application.

  The default is the values used for the previous plan.

8 INCLUDE PACKAGE LIST?
Lets you include a list of collections and packages in the plan. If you specify YES, a separate panel displays on which you must enter the package location, collection name, and package name for each package to include in the plan (see "Panels for entering lists of values" on page 909). This field can either add a package list to a plan that did not have one, or replace an existing package list.

You can specify a location name from 1 to 16 characters, a collection ID from 1 to 18 characters, and a package ID from 1 to 8 characters. Separate two or more package list parameters with a comma. If you specify a location name, it must be in the catalog table SYSIBM.LOCATIONS. The default location is the package list used for the previous plan.

9 CURRENT SERVER
Lets you specify the initial server to receive and process SQL statements in this plan. You can specify a name from 1 to 16 characters, which you must previously define in the catalog table SYSIBM.LOCATIONS.

If you specify a remote server, Db2 connects to that server when the first SQL statement executes. The default is the name of the local Db2 subsystem.

10 INCLUDE PATH?
Indicates which one of the following actions you want to perform:

- Request that Db2 uses the same schema names as when the plan was bound for resolving unqualified distinct type, user-defined function, and stored procedure names in SQL statements. Choose SAME to perform this action. This is the default.
- Supply a list of schema names that Db2 searches when it resolves unqualified distinct type, user-defined function, and stored procedure names in SQL statements. Choose YES to perform this action.
- Request that Db2 resets the SQL path to SYSIBM, SYSFUN, SYSPROC, and the plan owner. Choose DEFAULT to perform this action.

If you specify YES, Db2 displays a panel in which you specify the names of schemas for Db2 to search.

Related reference
Defaults for Bind Plan and Defaults for Rebind Plan panels
These DB2I panels let you change your defaults for BIND PLAN and REBIND PLAN options.

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Free Package panel

The DB2I Free Package panel is the first of two panels through which you can specify options for freeing an application package.

The following figure shows the free package options.

```
DSNEBP18                  FREE PACKAGE                        SSID: DSN
 COMMAND ===>_

  1  Free ALL packages ......... ===>     (* to free authorized packages)

 or
     Enter package name(s) to be freed:
  2  LOCATION NAME ............. ===>             (Defaults to local)
  3  COLLECTION-ID ............. ===>              > (Required)
  4  PACKAGE-ID ................ ===>     >(* to free all packages)
  5  VERSION-ID ................ ===>
                                           (*, Blank, (), or version-id)
  6  ADDITIONAL PACKAGES?....... ===>      (Yes to include more packages)
  7  PLAN MANAGEMENT SCOPE ..... ===> ALL        (ALL or INACTIVE)
```

Figure 60. The Free Package panel

This panel lets you specify options for erasing packages.

1 Free ALL packages
Lets you free (erase) all packages for which you have authorization or to which you have BINDAGENT authority. To do so, place an asterisk (*) in this field; otherwise, leave it blank.

2 LOCATION NAME
Lets you specify the location name of the DBMS to free the package. You can specify a name from 1 to 16 characters.

3 COLLECTION-ID
Lets you specify the collection from which you want to delete packages for which you own or have BINDAGENT privileges. You can specify a name from 1 to 128 characters, or an asterisk (*) to free all collections in the local Db2 system. You cannot use the asterisk to free a remote collection. This field is scrollable.

4 PACKAGE-ID
Lets you specify the name of the package to free. You can specify a name from 1 to 128 characters, or an asterisk (*) to free all packages in the specified collections in the local Db2 system. You cannot use the asterisk to free a remote package. The name you specify must be in the Db2 catalog tables. This field is scrollable.

5 VERSION-ID
Lets you specify the version of the package to free. You can specify an identifier from 1 to 64 characters, or an asterisk (*) to free all versions of the specified collections and packages in the local Db2 system. You cannot use the asterisk to free a remote version.

6 ADDITIONAL PACKAGES?
Lets you indicate whether to name more packages to free. Use YES to specify more packages on an additional panel, described in "Panels for entering lists of values" on page 909. The default is NO.

7 PLAN MANAGEMENT SCOPE
Specifies whether Db2 frees all copies of the package, or only the inactive previous and original copies. This value corresponds to the PLANMGMTSCOPE option. The default value is ALL.

Free Plan panel

The DB2I Free Plan panel is the first of two panels through which you can specify options for freeing an application plan.

Figure 61 on page 919 shows the free plan options.

```
DSNEBP09                 FREE PLAN                           SSID: DSN
 COMMAND ===>_

 Enter plan name(s) to be freed:
  1  PLAN NAME ............ ===>         (* to free all authorized plans)
  2  ADDITIONAL PLANS? .... ===>         (Yes to include more plans)
```

Figure 61. The Free Plan panel

This panel lets you specify options for freeing plans.

1 PLAN NAME
Lets you name the application plan to delete from Db2. Use an asterisk to free all plans for which you have BIND authority. You can specify a name from 1 to 8 characters, and the first character must be alphabetic.

If there are errors, the free process terminates for that plan and continues with the next plan.

2 ADDITIONAL PLANS?
Lets you indicate whether to name more plans to free. Use YES to specify more plans on an additional panel, described in "Panels for entering lists of values" on page 909. The default is NO.

Chapter 10. Running an application on Db2 for z/OS

You can run your application after you have processed the SQL statements, compiled and link-edited the application, and bound the application.

About this task

At run time, Db2 verifies that the information in the application plan and its associated packages is consistent with the corresponding information in the Db2 catalog. If any destructive changes, such as DROP or REVOKE, occur (either to the data structures that your application accesses or to the binder's authority to access those data structures), Db2 automatically rebinds packages or the plan as needed.

Establishing a test environment: This topic describes how to design a test data structure and how to fill tables with test data.

CICSBefore you run an application, ensure that the following two conditions are met:

- The corresponding entries in the SNT and RACF control areas authorize your application to run.
- The program and its transaction code are defined in the CICS CSD.

The system administrator is responsible for these functions.

DSN command processor

The DSN command processor is a TSO command processor that runs in TSO foreground or under TSO in a JES-initiated batch environment.

It uses the TSO attachment facility to access Db2. The DSN command processor provides an alternative method for running programs that access Db2 in a TSO environment.

When you run an application by using the DSN command processor, that application can run in a trusted connection if Db2 finds a matching trusted context.

You can use the DSN command processor implicitly during program development for functions such as:

- Using the declarations generator (DCLGEN)
- Running the BIND, REBIND, and FREE subcommands on Db2 plans and packages for your program
- Using SPUFI (SQL Processor Using File Input) to test some of the SQL functions in the program

The DSN command processor runs with the TSO terminal monitor program (TMP). Because the TMP runs in either foreground or background, DSN applications run interactively or as batch jobs.

The DSN command processor can provide these services to a program that runs under it:

- Automatic connection to Db2
- Attention key support
- Translation of return codes into error messages

Limitations of the DSN command processor

When using DSN services, your application runs under the control of DSN. Because TSO executes the ATTACH macro to start DSN, and DSN executes the ATTACH macro to start a part of itself, your application gains control that is two task levels below TSO.

Because your program depends on DSN to manage your connection to Db2:

- If Db2 is down, your application cannot begin to run.
- If Db2 terminates, your application also terminates.
- An application can use only one plan.

If these limitations are too severe, consider having your application use the call attachment facility or Resource Recovery Services attachment facility. For more information about these attachment facilities, see "Call attachment facility" on page 41 and "Resource Recovery Services attachment facility" on page 71.

DSN return code processing

At the end of a DSN session, register 15 contains the highest value that is placed there by any DSN subcommand that is used in the session or by any program that is run by the RUN subcommand. Your run time environment might format that value as a return code. However, the value does not originate in DSN.

Related concepts

TSO attachment facility (Introduction to Db2 for z/OS)

Related reference

DSN (TSO) (Db2 Commands)

Related information

Command types and environments in Db2 (Db2 Commands)

DB2I Run panel

The DB2I Run panel lets you start an application program that can contain SQL statements.

You can reach the Run panel only through the DB2I Primary Options Menu. You can accomplish the same task using the "Program Preparation: Compile, Link, and Run" panel. You should use this panel if you have already prepared the program and simply want to run it. Figure 62 on page 922 shows the run options.

```
DSNERP01                       RUN                            SSID: DSN
 COMMAND ===>_

 Enter the name of the program you want to run:
  1  DATA SET NAME ===>
  2  PASSWORD..... ===>        (Required if data set is password protected)

 Enter the following as desired:
  3  PARAMETERS .. ===>
  4  PLAN NAME ... ===>        (Required if different from program name)
  5  WHERE TO RUN  ===>        (FOREGROUND, BACKGROUND, or EDITJCL)
```

Figure 62. The Run panel

This panel lets you run existing application programs.

1 DATA SET NAME

Lets you specify the name of the partitioned data set that contains the load module. If the module is in a data set that the operating system can find, you can specify the member name only. There is no default.

If you do not enclose the name in apostrophes, a standard TSO prefix (user ID) and suffix (.LOAD) is added.

2 PASSWORD

Lets you specify the data set password if needed. The RUN processor does not check whether you need a password. If you do not enter a required password, your program does not run.

3 PARAMETERS

Lets you specify a list of parameters you want to pass either to your host language run time processor, or to your application. You should separate items in the list with commas, blanks, or both. You can leave this field blank.

Use a slash (/) to separate the options for your run time processor from those for your program.

- For PL/I and Fortran, run time processor parameters must appear on the left of the slash, and the application parameters must appear on the right.

```
run time processor parameters / application parameters
```

- For COBOL, reverse this order. run time processor parameters must appear on the right of the slash, and the application parameters must appear on the left.
- For assembler and C, there is no supported run time environment, and you need not use the slash to pass parameters to the application program.

4 PLAN NAME
Lets you specify the name of the plan to which the program is bound. The default is the member name of the program.

5 WHERE TO RUN
Lets you indicate whether to run in the foreground or background. You can also specify EDITJCL, in which case you are able to edit the job control statement before you run the program. Use:

FOREGROUND to immediately run the program in the foreground with the specified values.

BACKGROUND to create and immediately submit to run a file containing a DSNH CLIST using the JOB control statement from either DB2I Defaults Panel 2 or your site's SUBMIT exit. The program runs in the background.

EDITJCL to create and open a file containing a DSNH CLIST in edit mode. You can then submit the CLIST or save it. The program runs in the background.

Running command processors
To run a command processor (CP), use the following commands from the TSO ready prompt or as a TSO TMP:

```
DSN SYSTEM (Db2-subsystem-name)
RUN CP PLAN (plan-name)
```

The RUN subcommand prompts you for more input. The end the DSN processor, use the END command.

Running a program in TSO foreground

Use the DB2I RUN panel to run a program in TSO foreground. Alternatively, you can issue the DSN command, followed by the RUN subcommand of DSN.

About this task

Before running the program, be sure to allocate any data sets that your program needs.

The following example shows how to start a TSO foreground application. The name of the application is SAMPPGM, and *ssid* is the system ID:

```
TSO Prompt:    READY
Enter:         DSN SYSTEM(ssid)
DSN Prompt:    DSN
Enter:         RUN PROGRAM(SAMPPGM) -
               PLAN(SAMPLAN) -
               LIB(SAMPPROJ.SAMPLIB) -
               PARMS('/D01 D02 D03')
⋮ (Here the program runs and might prompt you for input)
DSN Prompt:    DSN
Enter:         END
TSO Prompt:    READY
```

This sequence also works in ISPF option 6. You can package this sequence in a CLIST. Db2 does not support access to multiple Db2 subsystems from a single address space.

The PARMS keyword of the RUN subcommand enables you to pass parameters to the run time processor and to your application program:

```
PARMS ('/D01, D02, D03')
```

The slash (/) indicates that you are passing parameters. For some languages, you pass parameters and run time options in the form PARMS('*parameters/run time-options*). An example of the PARMS keyword might be:

```
PARMS ('D01, D02, D03/')
```

Check your host language publications for the correct form of the PARMS option.

Running a Db2 REXX application

You run Db2 REXX applications under TSO. You do not precompile, compile, link-edit, or bind Db2 REXX applications before you run them.

About this task

In a batch environment, you might use statements like these to invoke application REXXPROG:

```
//RUNREXX EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSEXEC  DD DISP=SHR,DSN=SYSADM.REXX.EXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
%REXXPROG parameters
```

The SYSEXEC data set contains your REXX application, and the SYSTSIN data set contains the command that you use to invoke the application.

Invoking programs through the Interactive System Productivity Facility

You can use ISPF to invoke programs that connect to Db2 through the call attachment facility (CAF).

About this task

The ISPF/CAF sample connection manager programs (DSN8SPM and DSN8SCM) take advantage of the ISPLINK SELECT services, letting each routine make its own connection to Db2 and establish its own thread and plan.

With the same modular structure as in the previous example, using CAF is likely to provide greater efficiency by reducing the number of CLISTs. This does not mean, however, that any Db2 function executes more quickly.

Disadvantages: Compared to the modular structure using DSN, the structure using CAF is likely to require a more complex program, which in turn might require assembler language subroutines. For more information, see "Call attachment facility" on page 41.

ISPF

The Interactive System Productivity Facility (ISPF) helps you to construct and execute dialogs. Db2 includes a sample application that illustrates how to use ISPF through the call attachment facility (CAF).

Each scenario has advantages and disadvantages in terms of efficiency, ease of coding, ease of maintenance, and overall flexibility.

Using ISPF and the DSN command processor

There are some restrictions on how you make and break connections to Db2 in any structure. If you use the PGM option of ISPF SELECT, ISPF passes control to your load module by the LINK macro; if you use CMD, ISPF passes control by the ATTACH macro.

The DSN command processor permits only single task control block (TCB) connections. Take care not to change the TCB after the first SQL statement. ISPF SELECT services change the TCB if you started DSN under ISPF, so you cannot use these to pass control from load module to load module. Instead, use LINK, XCTL, or LOAD.

The following figure shows the task control blocks that result from attaching the DSN command processor below TSO or ISPF.

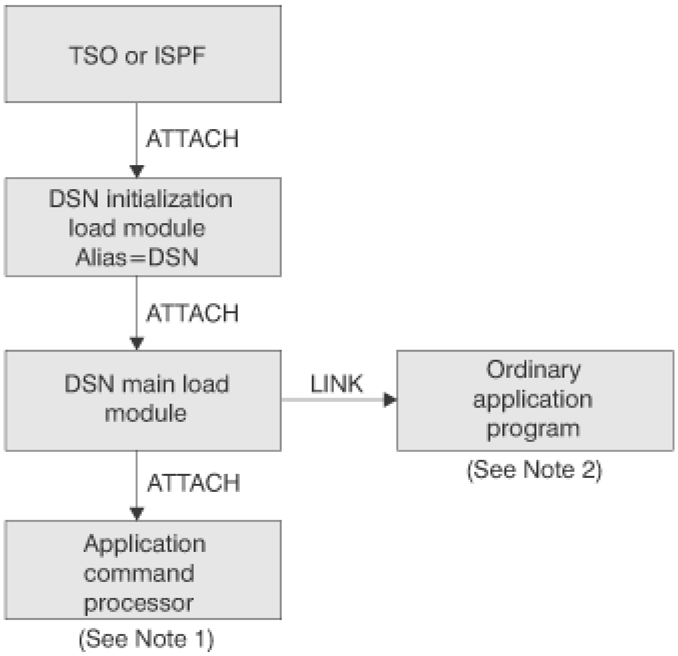


Figure 63. DSN task structure

Notes:

1. The RUN command with the CP option causes DSN to attach your program and create a new TCB.
2. The RUN command without the CP option causes DSN to link to your program.

If you are in ISPF and running under DSN, you can perform an ISPLINK to another program, which calls a CLIST. In turn, the CLIST uses DSN and another application. Each such use of DSN creates a separate unit of recovery (process or transaction) in Db2.

All such initiated DSN work units are unrelated, with regard to isolation (locking) and recovery (commit). It is possible to deadlock with yourself; that is, one unit (DSN) can request a serialized resource (a data page, for example) that another unit (DSN) holds incompatibly.

A COMMIT in one program applies only to that process. There is no facility for coordinating the processes.

Related concepts

Dynamic SQL and the ISPF/CAF application (Db2 Installation and Migration)

Printing options for the sample application listings (Db2 Installation and Migration)

Sample applications supplied with Db2 for z/OS
Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

DSN command processor

The DSN command processor is a TSO command processor that runs in TSO foreground or under TSO in a JES-initiated batch environment.

Invoking a single SQL program through ISPF and DSN

When you invoke a single SQL program through ISPF and DSN, you should first invoke ISPF, which displays the data and selection panels. When you select the program on the selection panel, ISPF calls a CLIST that runs the program.

About this task

A corresponding CLIST might contain:

```
DSN
    RUN PROGRAM(MYPROG) PLAN(MYPLAN)
END
```

The application has one large load module and one plan.

Disadvantages: For large programs of this type, you want a more modular design, making the plan more flexible and easier to maintain. If you have one large plan, you must rebind the entire plan whenever you change a module that includes SQL statements. To achieve a more modular construction when all parts of the program use SQL, consider using packages. See "Db2 program preparation overview" on page 881 You cannot pass control to another load module that makes SQL calls by using ISPLINK; rather, you must use LINK, XCTL, or LOAD and BALR.

If you want to use ISPLINK, then call ISPF to run under DSN:

```
DSN
    RUN PROGRAM(ISPF) PLAN(MYPLAN)
END
```

You then need to leave ISPF before you can start your application.

Furthermore, the entire program is dependent on Db2; if Db2 is not running, no part of the program can begin or continue to run.

Invoking multiple SQL programs through ISPF and DSN

You can break a large application into several different functions. Each function communicates through a common pool of shared variables, which is controlled by ISPF.

About this task

You might write some functions as separately compiled and loaded programs, others as EXECs or CLISTs. You can start any of those programs or functions through the ISPF SELECT service, and you can start that from a program, a CLIST, or an ISPF selection panel.

When you use the ISPF SELECT service, you can specify whether ISPF should create a new ISPF variable pool before calling the function. You can also break a large application into several independent parts, each with its own ISPF variable pool.

You can call different parts of the program in different ways. For example, you can use the PGM option of ISPF SELECT:

```
PGM(program-name) PARM(parameters)
```

Alternatively, you can use the CMD option:

```
CMD(command)
```

For a part that accesses Db2, the command can name a CLIST that starts DSN:

```
DSN
    RUN PROGRAM(PART1) PLAN(PLAN1) PARM(input from panel)
END
```

Breaking the application into separate modules makes it more flexible and easier to maintain. Furthermore, some of the application might be independent of Db2; portions of the application that do not call Db2 can run, even if Db2 is not running. A stopped Db2 database does not interfere with parts of the program that refer only to other databases.

Disadvantages: The modular application, on the whole, has to do more work. It calls several CLISTs, and each one must be located, loaded, parsed, interpreted, and executed. It also makes and breaks connections to Db2 more often than the single load module. As a result, you might lose some efficiency.

Loading and running a batch program

You can run a DL/I batch program by running module DSNMTV01, which loads your application, or by running the application program directly.

Procedure

To run a program using Db2, you need a Db2 plan.

The bind process creates the Db2 plan. Db2 first verifies whether the DL/I batch job step can connect to Db2. Then Db2 verifies whether the application program can access Db2 and enforce user identification of batch jobs accessing Db2.

The two ways to submit DL/I batch applications to Db2 are:

- DSNMTV01 can be specified as the application program name for the batch region. When this method is used, control is given to DSNMTV01. When the Db2 environment is established, control is passed to the application program.
- The application program name can be specified for the batch region. Control is given to DSNMTV01 directly to establish the external subsystem environment for Db2. When the Db2 environment is established, control is passed to the application program that is specified in the batch region. To accomplish this, in the batch region startup procedure in your application JCL, specify the following information:
  - MBR=*application-name*
  - SSM=*DB2-subsystem-name*

Examples

Example: Submitting a DL/I batch application without using DSNMTV01

The skeleton JCL in the following example illustrates a COBOL application program, IVP8CP22, that runs using Db2 DL/I batch support.

```
//TEPCTEST JOB 'USER=ADMF001',MSGCLASS=A,MSGLEVEL=(1,1),
//          TIME=1440,CLASS=A,USER=SYSADM,PASSWORD=SYSADM
//*******************************
//BATCH  EXEC DLIBATCH,PSB=IVP8CA,MBR=IVP8CP22,
//       BKO=Y,DBRC=N,IRLM=N,SSM=SSDQ
//*******************************
//SYSPRINT DD SYSOUT=A
//REPORT   DD SYSOUT=*
//G.DDOTV02 DD DSN=&TEMP,DISP=(NEW,PASS,DELETE),
//    SPACE=(CYL,(10,1),RLSE),
//    UNIT=SYSDA,DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
SSDQ,SYS1,DSNMIN10,,Q,",DSNMTES1,,IVP8CP22
//G.SYSIN DD *
/*
//****************************************************
//*   ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET
//****************************************************
```

```
//PRTLOG EXEC PGM=DFSERA10,COND=EVEN
//STEPLIB   DD DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSUT1    DD DSN=&TEMP,DISP=(OLD,DELETE)
//SYSIN     DD *
CONTROL  CNTL K=000,H=8000
OPTION   PRINT
/*
```

Example: Submitting a DL/I batch application using DSNMTV01

The following skeleton JCL example illustrates a COBOL application program, IVP8CP22, that runs using Db2 DL/I batch support.

- The first step uses the standard DLIBATCH IMS procedure.
- The second step shows how to use the DFSERA10 IMS program to print the contents of the DDOTV02 output data set.

```
//ISOCS04  JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
//        MSGCLASS=T,CLASS=A
//JOBLIB  DD  DISP=SHR,
//            DSN=prefix.SDSNLOAD
//* ******************************************************************
//*
//* THE FOLLOWING STEP SUBMITS COBOL JOB IVP8CP22, WHICH UPDATES
//* BOTH DB2 AND DL/I DATABASES.
//*
//* ******************************************************************
//UPDTE    EXEC DLIBATCH,DBRC=Y,LOGT=SYSDA,COND=EVEN,
//  MBR=DSNMTV01,PSB=IVP8CA,BKO=Y,IRLM=N
//G.STEPLIB  DD
//           DD DSN=prefix.SDSNLOAD,DISP=SHR
//           DD  DSN=prefix.RUNLIB.LOAD,DISP=SHR
//           DD  DSN=CEE.SCEERUN,DISP=SHR
//           DD  DSN=IMS.PGMLIB,DISP=SHR
//G.DDOTV02  DD  DSN=&TEMP1,DISP=(NEW,PASS,DELETE),
//         SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
//         DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02  DD  *
  SSDQ,SYS1,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//***************************************************************
//***   ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET      ***
//***************************************************************
//STEP3     EXEC  PGM=DFSERA10,COND=EVEN
//STEPLIB   DD    DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSNAME=&TEMP1,DISP=(OLD,DELETE)
//SYSIN     DD    *
CONTROL  CNTL K=000,H=8000
OPTION   PRINT
/*
//
```

Related concepts

Input and output data sets for DL/I batch jobs
DL/I batch jobs require an input data set with DD name DDITV02 and an output data set with DD name DDOTV02.

Authorization for running a batch DL/I program

When a DL/I batch application tries to run the first SQL statement, Db2 checks whether the authorization ID has the EXECUTE privilege for the plan. Db2 uses the same ID for subsequent authorization checks and also identifies records from the accounting and performance traces.

The primary authorization ID is the value of the USER parameter on the job statement, if that is available. If that parameter is not available, the primary authorization ID is the TSO logon name if the job is submitted. Otherwise, the primary authorization ID is the IMS PSB name. In that case, however, the ID must not begin with the string "SYSADM" because this string causes the job to abnormally terminate. The batch job is rejected if you try to change the authorization ID in an exit routine.

Restarting a batch program

To restart a batch program that updates data, first run the IMS Batch Backout utility, followed by a restart job indicating the last successful checkpoint ID.

About this task

For guidelines on finding the last successful checkpoint, see "Finding the DL/I batch checkpoint ID" on page 930.

Examples

Example: Batch checkout with JCL

The skeleton JCL example that follows illustrates a batch backout for PSB=IVP8CA.

```
//ISOCS04  JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
//        MSGCLASS=T,CLASS=A
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//*                                                           *
//*  BACKOUT TO LAST CHKPT.                                   *
//*                 IF RC=0028 LOG WITH NO-UPDATE             *
//*                                                           *
//* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
//BACKOUT  EXEC PGM=DFSRRC00,
//        PARM='DLI,DFSBBO00,IVP8CA,,,,,,,,,,Y,N,,Y',
//         REGION=2600K,COND=EVEN            |
//*                                          ---> DBRC ON
//STEPLIB DD  DSN=IMS.RESLIB,DISP=SHR
//IMS          DD   DSN=IMS.PSBLIB,DISP=SHR
//             DD   DSN=IMS.DBDLIB,DISP=SHR
//*
//* IMSLOGR  DD   data set is required
//* IEFRDER  DD   data set is required
//DFSVSAMP DD  *
OPTIONS,LTWA=YES
2048,7
1024,7
/*
//SYSIN DD DUMMY
/*
```

Example: restarting a DL/I batch job with JCL

Operational procedures can restart a DL/I batch job step for an application program using IMS XRST and symbolic CHKP calls.

You cannot restart a BMP application program in a Db2 DL/I batch environment. The symbolic checkpoint records are not accessed, causing an IMS user abend U0102.

To restart a batch job that terminated abnormally or prematurely, find the checkpoint ID for the job on the z/OS system log or from the SYSOUT listing of the failing job. Before you restart the job step, place the checkpoint ID in the CKPTID=value option of the DLIBATCH procedure, submit the job. If the default connection name is used (that is, you did not specify the connection name option in the DDITV02 input data set), the job name of the restart job must be the same as the failing job. Refer to the following skeleton example, in which the last checkpoint ID value was IVP80002:

```
//ISOCS04  JOB 3000,OJALA,MSGLEVEL=(1,1),NOTIFY=OJALA,
//        MSGCLASS=T,CLASS=A
//* ******************************************************************
//*
//* THE FOLLOWING STEP RESTARTS COBOL PROGRAM IVP8CP22, WHICH UPDATES
//* BOTH DB2 AND DL/I DATABASES, FROM CKPTID=IVP80002.
//*
//* ******************************************************************
//RSTRT    EXEC DLIBATCH,DBRC=Y,COND=EVEN,LOGT=SYSDA,
//  MBR=DSNMTV01,PSB=IVP8CA,BKO=Y,IRLM=N,CKPTID=IVP80002
//G.STEPLIB  DD
//         DD
//         DD  DSN=prefix.SDSNLOAD,DISP=SHR
//         DD  DSN=prefix.RUNLIB.LOAD,DISP=SHR
//         DD  DSN=SYS1.COB2LIB,DISP=SHR
//         DD  DSN=IMS.PGMLIB,DISP=SHR
//*        other program libraries
```

```
//* G.IEFRDER  data set required
//* G.IMSLOGR  data set required
//G.DDOTV02  DD  DSN=&TEMP2,DISP=(NEW,PASS,DELETE),
//         SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
//         DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02  DD  *
  DB2X,SYS1,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//****************************************************************
//***  ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET       ***
//****************************************************************
//STEP8     EXEC  PGM=DFSERA10,COND=EVEN
//STEPLIB   DD    DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSNAME=&TEMP2,DISP=(OLD,DELETE)
//SYSIN     DD    *
CONTROL  CNTL K=000,H=8000
OPTION   PRINT
/*
//
```

Finding the DL/I batch checkpoint ID

When an application program issues an IMS CHKP call, IMS sends the checkpoint ID to the z/OS console and the SYSOUT listing in message DFS0540I.

About this task

IMS also records the checkpoint ID in the type X'41' IMS log record. Symbolic CHKP calls also create one or more type X'18' records on the IMS log. XRST uses the type X'18' log records to reposition DL/I databases and return information to the application program.

During the commit process the application program checkpoint ID is passed to Db2. If a failure occurs during the commit process, creating an indoubt work unit, Db2 remembers the checkpoint ID. You can use the following techniques to find the last checkpoint ID:

- Look at the SYSOUT listing for the job step to find message DFS0540I, which contains the checkpoint IDs that are issued. Use the last listed checkpoint ID.
- Look at the z/OS console log to find message DFS0540I that contains the checkpoint ID that is issued for this batch program. Use the last listed checkpoint ID.
- Submit the IMS Batch Backout utility to back out the DL/I databases to the last (default) checkpoint ID. When the batch backout finishes, message DFS395I provides the last valid IMS checkpoint ID. Use this checkpoint ID on restart.
- When restarting Db2, issue the command `-DISPLAY THREAD(*) TYPE(INDOUBT)` to obtain a possible indoubt unit of work (connection name and checkpoint ID). If you restarted the application program from this checkpoint ID, the program could work because the checkpoint is recorded on the IMS log; however, the program could fail with an IMS user abend U102 because IMS did not finish logging the information before the failure. In that case, restart the application program from the previous checkpoint ID.

Db2 performs one of two actions automatically when restarted, if the failure occurs outside the indoubt period: it either backs out the work unit to the prior checkpoint, or it commits the data without any assistance. If the operator then issues the following command, no work unit information is displayed:

```
-DISPLAY THREAD(*) TYPE(INDOUBT)
```

Running stored procedures from the command line processor

As an alternative to calling a stored procedure from an application program, you can use the command line processor to invoke stored procedures.

Procedure

To run a stored procedure from the command line processor:

1. Invoke the command line processor and connect to the appropriate Db2 subsystem. For more information about how to perform these tasks, see The Db2 command line processor (Db2 Commands).
2. Specify the CALL statement in the form that is acceptable for the command line processor.

Related tasks

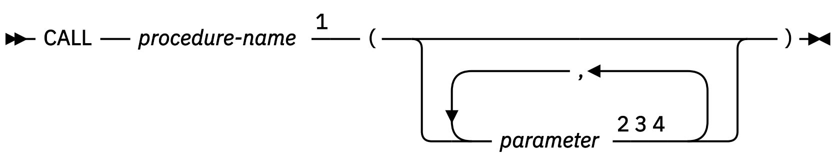
Calling a stored procedure from your application
To run a stored procedure, you can either call it from a client program or invoke it from the command line processor.

Implementing Db2 stored procedures (Db2 Administration Guide)

Command line processor CALL statement

Use the command line processor CALL statement to invoke stored procedures from the command line processor.

Use the following syntax for the command line processor CALL statement.



Notes:

[1] If you specify an unqualified stored procedure name, Db2 searches the schema list in the CURRENT PATH special register. Db2 searches this list for a stored procedure with the specified number of input and output parameters.

[2] Specify a question mark (?) as a placeholder for each output parameter.

[3] For non-numeric, BLOB, or CLOB input parameters, enclose each value in single quotation marks ('). The exception is if the data is a BLOB or CLOB value that is to be read from a file. In that case, use the notation `file://`*`fully qualified file name`*.

[4] Specify the input and output parameters in the order that they are specified in the signature for the stored procedure.

Examples

Example1

Assume that the TEST.DEPT_MEDIAN stored procedure was created with the following statement:

```
CREATE PROCEDURE TEST.DEPT_MEDIAN
(IN DEPTNUMBER SMALLINT, OUT MEDIANSALARY INT)
```

To invoke the stored procedure from the command line processor, you can specify the following CALL statement:

```
CALL TEST.DEPT_MEDIAN(51, ?)
```

Assume that the stored procedure returns a value of 25,000. The following information is displayed by the command line processor:

```
Value of output parameters
--------------------------
Parameter Name : MEDIANSALARY
Parameter Value : 25000
```

Example 2

Suppose that stored procedure TEST.BLOBSP is defined with one input parameter of type BLOB and one output parameter. You can invoke this stored procedure from the command line processor with the following statement:

```
CALL TEST.BLOBSP(file:///tmp/photo.bmp,?)
```

The command line processor reads the contents from /tmp/photo.bmp as the input parameter. Alternatively, you can invoke this stored procedure by specifying the input parameter in the CALL statement itself, as in the following example:

```
CALL TEST.BLOBSP('abcdef',?)
```

Example of running a batch Db2 application in TSO

Most application programs that are written for the batch environment run under the TSO Terminal Monitor Program (TMP) in background mode.

The following figure shows the JCL statements that you need in order to start such a job. The list that follows explains each statement.

```
//jobname JOB USER=MY DB2ID
//GO EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=prefix.SDSNEXIT,DISP=SHR
//        DD DSN=prefix.SDSNLOAD,DISP=SHR
⋮
//SYSTSPRT DD SYSOUT=A
//SYSTSIN DD *
 DSN SYSTEM (ssid)
 RUN PROG (SAMPPGM) -
   PLAN (SAMPLAN) -
   LIB (SAMPPROJ.SAMPLIB) -
   PARMS ('/D01 D02 D03')
 END
 /*
```

- The JOB option identifies this as a job card. The USER option specifies the Db2 authorization ID of the user.
- The EXEC statement calls the TSO Terminal Monitor Program (TMP).
- The STEPLIB statement specifies the library in which the DSN Command Processor load modules and DSNHDECP or a user-specified application defaults module reside. It can also reference the libraries in which user applications, exit routines, and the customized DSNHDECP module reside. The customized DSNHDECP module is created during installation.
- Subsequent DD statements define additional files that are needed by your program.
- The DSN command connects the application to a particular Db2 subsystem.
- The RUN subcommand specifies the name of the application program to run.
- The PLAN keyword specifies plan name.
- The LIB keyword specifies the library that the application should access.
- The PARMS keyword passes parameters to the run time processor and the application program.
- END ends the DSN command processor.

Usage notes

- Keep DSN job steps short.
- **Recommendation:** Do not use DSN to call the EXEC command processor to run CLISTs that contain ISPEXEC statements; results are unpredictable.
- If your program abends or gives you a non-zero return code, DSN terminates.
- You can use a group attachment or subgroup attachment name instead of a specific *ssid* to connect to a member of a data sharing group.

Related tasks

Running TSO application programs (Db2 Administration Guide)

Related reference

Executing the terminal monitor program (TSO/E Customization)

DSN (TSO) (Db2 Commands)

Example of calling applications in a command procedure

As an alternative to foreground or batch calls to an application, you can run a TSO or batch application by using a command procedure (CLIST).

The following CLIST calls a Db2 application program named MYPROG. *ssid* represents the Db2 subsystem name, or group attachment or subgroup attachment name.

```
PROC 0                         /* INVOCATION OF DSN FROM A CLIST      */
  DSN SYSTEM(ssid)             /* INVOKE DB2 SUBSYSTEM ssid           */
  IF &LASTCC = 0 THEN          /* BE SURE DSN COMMAND WAS SUCCESSFUL  */
    DO                         /* IF SO THEN DO DSN RUN SUBCOMMAND    */
      DATA                     /* ELSE OMIT THE FOLLOWING:            */
        RUN PROGRAM(MYPROG)
        END
      ENDDATA                  /* THE RUN AND THE END ARE FOR DSN     */
    END
EXIT
```

IMS: To run a message-driven program

First, ensure that you can respond to the program's interactive requests for data and that you can recognize the expected results. Then, enter the transaction code that is associated with the program. Users of the transaction code must be authorized to run the program.

To run a non-message-driven program

CICSTo run a program

First, ensure that the corresponding entries in the SNT and RACF control areas allow run authorization for your application. The system administrator is responsible for these functions.

Submit the job control statements that are needed to run the program.

Also, be sure to define to CICS the transaction code that is assigned to your program and the program itself.

Make a new copy of the program

Issue the NEWCOPY command if CICS has not been reinitialized since the program was last bound and compiled.

Chapter 11. Testing and debugging an application program on Db2 for z/OS

Depending on the situation, testing your application program might involve setting up a test environment, testing SQL statements, debugging your programs, and reading output from the precompiler.

Related tasks

Modeling a production environment on a test subsystem (Db2 Performance)

Modeling your production system statistics in a test subsystem (Db2 Performance)

Designing a test data structure

When you test an application that accesses Db2 data, you should have Db2 data available for testing. To do this, you can create test tables and views.

About this task

- **Test views of existing tables:** If your application does not change a set of Db2 data and the data exists in one or more production-level tables, you might consider using a view of existing tables.
- **Test tables:** To create a test table, you need a database and table space. Talk with your DBA to make sure that a database and table spaces are available for your use.

  If the data that you want to change already exists in a table, consider using the LIKE clause of CREATE TABLE. If you want others besides yourself to have ownership of a table for test purposes, you can specify a secondary ID as the owner of the table. You can do this with the SET CURRENT SQLID statement.

If your location has a separate Db2 system for testing, you can create the test tables and views on the test system. This information assumes that you do all testing on a separate system, and that the person who created the test tables and views has an authorization ID of TEST. The table names are TEST.EMP, TEST.PROJ and TEST.DEPT.

Related concepts

Authorization IDs (Managing Security)

Related tasks

Modeling a production environment on a test subsystem (Db2 Performance)

Modeling your production system statistics in a test subsystem (Db2 Performance)

Related reference

SET CURRENT SQLID (Db2 SQL)

Analyzing application data needs

To design tests of an application, you need to determine the type of data that the application uses and how the application accesses that data.

About this task

This information assumes that you do all testing on a separate system, and that the person who created the test tables and views has an authorization ID of TEST. The table names are TEST.EMP, TEST.PROJ and TEST.DEPT.

To design test tables and views, first analyze the data needs of your application.

Procedure

To analyze the data needs of your application:

1. List the data that your application accesses and describe how it accesses each data item.

   For example, suppose that you are testing an application that accesses the DSN8B10.EMP, DSN8B10.DEPT, and DSN8B10.PROJ tables. You might record the information about the data as shown in Table 152 on page 936.

Table 152. Description of the application data

Table or view name	Insert rows?	Delete rows?	Column name	Data type	Update access?
DSN8B10.EMP	No	No	EMPNO	CHAR(6)	No
			LASTNAME	VARCHAR(15)	No
			WORKDEPT	CHAR(3)	Yes
			PHONENO	CHAR(4)	Yes
			JOB	DECIMAL(3)	Yes
DSN8B10.DEPT	No	No	DEPTNO	CHAR(3)	No
			MGRNO	CHAR (6)	No
DSN8B10.PROJ	Yes	Yes	PROJNO	CHAR(6)	No
			DEPTNO	CHAR(3)	Yes
			RESPEMP	CHAR(6)	Yes
			PRSTAFF	DECIMAL(5,2)	Yes
			PRSTDATE	DECIMAL(6)	Yes
			PRENDATE	DECIMAL(6)	Yes

2. Determine the test tables and views that you need to test your application.

   Create a test table on your list when either of the following conditions exists:

   - The application modifies data in the table.
   - You need to create a view that is based on a test table because your application modifies data in the view.

   To continue the example, create these test tables:

   - TEST.EMP, with the following format:

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOB
⋮	⋮	⋮	⋮	⋮

   - TEST.PROJ, with the same columns and format as DSN8B10.PROJ, because the application inserts rows into the DSN8B10.PROJ table.

   To support the example, create a test view of the DSN8B10.DEPT table.

   - TEST.DEPT view, with the following format:

DEPTNO	MGRNO
⋮	⋮

   Because the application does not change any data in the DSN8B10.DEPT table, you can base the view on the table itself (rather than on a test table). However, a safer approach is to have a complete set of test tables and to test the program thoroughly using only test data.

Authorization for test tables and applications

Before you can create a table, you need to be authorized to create tables and to use the table space in which the table is to reside. You must also have authority to bind and run programs that you want to test.

Your DBA can grant you the necessary authorization to create and access tables and to bind and run programs.

If you intend to use existing tables and views (either directly or as the basis for a view), you need privileges to access those tables and views. Your DBA can grant those privileges.

To create a view, you must have authorization for each table and view on which you base the view. You then have the same privileges over the view that you have over the tables and views on which you based the view. Before trying the examples, have your DBA grant you the privileges to create new tables and views and to access existing tables. Obtain the names of tables and views that you are authorized to access (as well as the privileges you have for each table) from your DBA.

Example SQL statements to create a comprehensive test structure

You need to create a storage group, database, table space, and table to use as a test structure for your SQL application.

The following SQL statements show how to create a complete test structure to contain a small table named SPUFINUM. The test structure consists of:

- A storage group named SPUFISG
- A database named SPUFIDB
- A table space named SPUFITS in database SPUFIDB and using storage group SPUFISG
- A table named SPUFINUM within the table space SPUFITS

```
CREATE STOGROUP SPUFISG
   VOLUMES (user-volume-number)
   VCAT DSNCAT ;

CREATE DATABASE SPUFIDB ;

CREATE TABLESPACE SPUFITS
   IN SPUFIDB
   USING STOGROUP SPUFISG ;

CREATE TABLE SPUFINUM
   ( XVAL CHAR(12) NOT NULL,
     ISFLOAT FLOAT,
     DEC30 DECIMAL(3,0),
     DEC31 DECIMAL(3,1),
     DEC32 DECIMAL(3,2),
     DEC33 DECIMAL(3,3),
     DEC10 DECIMAL(1,0),
     DEC11 DECIMAL(1,1),
     DEC150 DECIMAL(15,0),
     DEC151 DECIMAL(15,1),
     DEC1515 DECIMAL(15,15) )
   IN SPUFIDB.SPUFITS ;
```

Related reference

CREATE DATABASE (Db2 SQL)
CREATE STOGROUP (Db2 SQL)
CREATE TABLE (Db2 SQL)
CREATE TABLESPACE (Db2 SQL)

Populating the test tables with data

To populate test tables, use SQL INSERT statements or the LOAD utility.

About this task

You can put test data into a table in several ways:

- INSERT ... VALUES (an SQL statement) puts one row into a table each time the statement executes.
- INSERT ... SELECT (an SQL statement) obtains data from an existing table (based on a SELECT clause) and puts it into the table that is identified in the INSERT statement.
- MERGE (an SQL statement) puts new data into a table and updates existing data.
- The LOAD utility obtains data from a sequential file (a non-Db2 file), formats it for a table, and puts it into a table.
- The Db2 sample UNLOAD program (DSNTIAUL) can unload data from a table or view and build control statements for the LOAD utility.
- The UNLOAD utility can unload data from a table and build control statements for the LOAD utility.

Related concepts

Sample applications supplied with Db2 for z/OS
Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

Related tasks

Inserting rows by using the INSERT statement
One way to insert data into tables is to use the SQL INSERT statement. This method is useful for inserting small amounts of data or inserting data from another table or view.

Inserting rows into a table from another table
You can copy one or more rows from one table into another table.

Inserting data and updating data in a single operation
You can update existing data and insert new data in a single operation. This operation is useful when you want to update a table with a set of rows, some of which are changes to existing rows and some of which are new rows.

Related reference

LOAD (Db2 Utilities)
UNLOAD (Db2 Utilities)

Methods for testing SQL statements

You can test your SQL statements by using SQL Processing Using File Input (SPUFI) or the command line processor.

Test with SPUFI: You can use SPUFI (an interface between ISPF and Db2) to test SQL statements in a TSO/ISPF environment. With SPUFI panels, you can put SQL statements into a data set that Db2 subsequently executes. The SPUFI Main panel has several functions that enable you to:

- Name an input data set to hold the SQL statements that are passed to Db2 for execution
- Name an output data set to contain the results of executing the SQL statements
- Specify SPUFI processing options

Test with the command line processor: You can use the command line processor to test SQL statements from UNIX System Services on z/OS.

SQL statements that are executed under SPUFI or the command line processor operate on actual tables (in this case, the tables that you created for testing). Consequently, before you access Db2 data:

- Make sure that all tables and views that your SQL statements refer to exist.
- If the tables or views do not exist, create them (or have your database administrator create them). You can use SPUFI or the command line processor to issue the CREATE statements that are used to create the tables and views that you need for testing.

Related concepts

The Db2 command line processor (Db2 Commands)

Related tasks

Executing SQL by using SPUFI
You can execute SQL statements dynamically in a TSO session by using the SPUFI (SQL processor using file input) facility.

Executing SQL by using SPUFI

You can execute SQL statements dynamically in a TSO session by using the SPUFI (SQL processor using file input) facility.

Before you begin

Before you use SPUFI, allocate an input data set to store the SQL statements that you want to execute, if such a data set does not already exist.

Before you begin this task, you can specify whether TSO message IDs are displayed by using the TSO PROFILE command. To view message IDs, type `TSO PROFILE MSGID` on the ISPF command line. To suppress message IDs, type `TSO PROFILE NOMSGID`.

These instructions assume that ISPF is available to you.

About this task

Important: Ensure that the TSO terminal CCSID matches the Db2 CCSID. If these CCSIDs do not match, data corruption can occur. If SPUFI issues the warning message DSNE345I, terminate your SPUFI session and notify the system administrator.

SPUFI can execute SQL statements that retrieve Unicode UTF-16 graphic data. However, SPUFI might not be able to display some characters, if those characters have no mapping in the target SBCS EBCDIC CCSID.

Procedure

To execute SQL by using SPUFI:

1. Open SPUFI and specify the initial options. To open SPUFI and specify initial options:
   a) Select SPUFI from the DB2I Primary Option Menu as shown in The DB2I primary option menu (Introduction to Db2 for z/OS).
      The SPUFI panel is displayed.
   b) Specify the input data set name and output data set name.
      An example of a SPUFI panel in which an input data set and output data set have been specified is shown in the following figure.

```
DSNESP01                            SPUFI                     SSID: DSN
===>
Enter the input data set name:   (Can be sequential or partitioned)
 1 DATA SET NAME..... ===> EXAMPLES(XMP1)
 2 VOLUME SERIAL..... ===>       (Enter if not cataloged)
 3 DATA SET PASSWORD. ===>       (Enter if password protected)

Enter the output data set name:   (Must be a sequential data set)
  4 DATA SET NAME..... ===> RESULT

Specify processing options:
  5 CHANGE DEFAULTS... ===> Y     (Y/N - Display SPUFI defaults panel?)
  6 EDIT INPUT........ ===> Y     (Y/N - Enter SQL statements?)
  7 EXECUTE........... ===> Y     (Y/N - Execute SQL statements?)
  8 AUTOCOMMIT........ ===> Y     (Y/N - Commit after successful run?)
  9 BROWSE OUTPUT..... ===> Y     (Y/N - Browse output data set?)

 For remote SQL processing:
 10 CONNECT LOCATION   ===>


 PRESS: ENTER to process      END to exit      HELP for more information
```

Figure 64. The SPUFI panel filled in

c) Specify new values in any of the other fields on the SPUFI panel.

   For more information about these fields, see "The SPUFI panel" on page 943.

2. Optional: Change the SPUFI defaults, as described in "Changing SPUFI defaults" on page 944.
3. Enter SQL statements in SPUFI.

   If the input data set that you specified on the SPUFI panel already contains all of the SQL statements that you want to execute, you can bypass this editing step by specifying NO for the EDIT INPUT field on the SPUFI panel. To enter SQL statements by using SPUFI:

   a) If the EDIT panel is not already open, on the SPUFI panel, specify Y in the EDIT INPUT field and press ENTER. If the input data set that you specified is empty, an empty EDIT panel opens. Otherwise, if the input data set contained SQL statements, those SQL statements are displayed in an EDIT panel.

   b) On the EDIT panel, use the ISPF EDIT program to enter or edit any SQL statements that you want to execute. Move the cursor to the first blank input line, and enter the first part of an SQL statement.

   If the input data set that you specified on the SPUFI panel already contains all of the SQL statements that you want to execute, you can bypass this editing step by specifying NO for the EDIT INPUT field on the SPUFI panel.

   You can enter the rest of the SQL statement on subsequent lines, as shown in the following figure:

```
EDIT --------userid.EXAMPLES(XMP1) --------------------- COLUMNS 001 072
COMMAND INPUT ===> SAVE                       SCROLL ===> PAGE
**********************************  TOP OF DATA  ***********************
000100 SELECT LASTNAME, FIRSTNME, PHONENO
000200   FROM DSN8B10.EMP
000300   WHERE WORKDEPT= 'D11'
000400   ORDER BY LASTNAME;
*********************************  BOTTOM OF DATA  *********************
```

Figure 65. The edit panel: After entering an SQL statement

Consider the following rules and recommendations when editing this input data set:

- Indent your lines and enter your statements on several lines to make your statements easier to read. Entering your statements on multiple lines does not change how your statements are processed.
- Do not put more than one SQL statement on a single line. If you do, the first statement executes, but Db2 ignores the other SQL statements on the same line. You can put more than one SQL statement in the input data set. Db2 executes the statements in the order in which you placed them in the data set.

- If the length of an SQL statement is greater than 71 bytes for an input data set with record length 79, or 72 bytes for an input data set with record length 80, you need to continue the SQL statement on additional lines of the SPUFI input data set. SPUFI concatenates the text on multiple lines without adding extra spaces at the end of any line. Therefore, if an SQL statement contains two values with a space between them, and the first value ends at the last allowed input position (71 or 72), you need to add an extra space on the next line before the second value.

  For example, suppose that the record length of your SPUFI input data set is 80, so the maximum length of an input line is 72. Also suppose that the SQL statement that you wish to enter is 81 bytes long. Bytes 69 through 81 contain `FROM MYTABLE;`. If you split the SQL statement after `FROM`, the first line of the statement ends in column 72, so you need to include a space in column 1 of the next line, before `MYTABLE;`. Otherwise, when SPUFI concatenates the two lines, the result is `FROMMYTABLE;`. When SPUFI runs the SQL statement, the SQL statement fails with SQLCODE -104.
- End each SQL statement with the statement terminator that you specified on the CURRENT SPUFI DEFAULTS panel.
- Save the data set every 10 minutes or so by entering the SAVE command.

c) Press the END PF key.

The data set is saved, and the SPUFI panel is displayed.

4. Process SQL statements with SPUFI.

You can use SPUFI to submit the SQL statements in a data set to Db2. To process SQL statements by using SPUFI:

a) On the SPUFI panel, specify YES in the EXECUTE field.

b) If you did not just finish using the EDIT panel to edit the input data set as described in "Entering SQL statements in SPUFI," specify NO In the EDIT INPUT field.

c) Press Enter.

SPUFI passes the input data set to Db2 for processing. Db2 executes the SQL statement in the input data set and sends the output to the output data set.

The output data set opens.

Your SQL statement might take a long time to execute, depending on how large a table Db2 must search, or on how many rows Db2 must process. In this case, you can interrupt the processing by pressing the PA1 key. Then respond to the message that asks you if you really want to stop processing. This action cancels the executing SQL statement. Depending on how much of the input data set Db2 was able to process before you interrupted its processing, Db2 might not have opened the output data set yet, or the output data set might contain all or part of the results data that are produced so far.

Results

SQL statements that exceed resource limit thresholds

Your system administrator might use the Db2 resource limit facility (governor) to set time limits for processing SQL statements in SPUFI. Those limits can be error limits or warning limits.

If you execute an SQL statement through SPUFI that runs longer than this error time limit, SPUFI terminates processing of that SQL statement and all statements that follow in the SPUFI input data set. SPUFI displays a panel that lets you commit or roll back the previously uncommitted changes that you have made. That panel is shown in the following figure.

```
DSNESP04          SQL STATEMENT RESOURCE LIMIT EXCEEDED       SSID: DSN
 ===>

 The following SQL statement has encountered an SQLCODE of -905 or -495:

Statement text







 Your SQL statement has exceeded the resource utilization threshold set
 by your site administrator.

 You must ROLLBACK or COMMIT all the changes made since the last COMMIT.
 SPUFI processing for the current input file will terminate immediately
 after the COMMIT or ROLLBACK is executed.

  1  NEXT ACTION ===>           (Enter COMMIT or ROLLBACK)

 PRESS:  ENTER to process                             HELP for more information
```

Figure 66. The resource limit facility error panel

If you execute an SQL statement through SPUFI that runs longer than the warning time limit for predictive governing, SPUFI displays the SQL STATEMENT RESOURCE LIMIT EXCEEDED panel. On this panel, you can tell Db2 to continue executing that statement, or stop processing that statement and continue to the next statement in the SPUFI input data set. That panel is shown in the following figure.

```
DSNESP05          SQL STATEMENT RESOURCE LIMIT EXCEEDED       SSID: DSN
 ===>

 The following SQL statement has encountered an SQLCODE of 495:

Statement text







 You can now either CONTINUE executing this statement or BYPASS the execution
 of this statement. SPUFI processing for the current input file will continue
 after the CONTINUE or BYPASS processing is completed.

  1  NEXT ACTION ===>           (Enter CONTINUE or BYPASS)

 PRESS:  ENTER to process                             HELP for more information
```

Figure 67. The resource limit facility warning panel

Related tasks

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Related reference

Using ISPF/PDF to Allocate Data Sets (z/OS ISPF User's Guide Vol II)

Related information

Lesson 1.1: Querying data interactively (Introduction to Db2 for z/OS)

Content of a SPUFI input data set

A SPUFI input data set can contain SQL statements, comments, and SPUFI control statements.

You can put comments about SQL statements either on separate lines or on the same line. In either case, use two hyphens (--) to begin a comment. Specify any text other than #SET TERMINATOR or #SET TOLWARN after the comment marker. Db2 ignores everything else to the right of the two hyphens.

The SPUFI panel

The SPUFI panel is the first panel that you need to fill out to run the SPUFI application.

After you complete any fields on the SPUFI panel and press Enter, those settings are saved. When the SPUFI panel displays again, the data entry fields on the panel contain the values that you previously entered. You can specify data set names and processing options each time the SPUFI panel is displayed, as needed. Values that you do not change remain in effect.

The following descriptions explain the fields that are available on the SPUFI panel.

1,2,3 INPUT DATA SET NAME
Identify the input data set in fields 1 through 3. This data set contains one or more SQL statements that you want to execute. Allocate this data set before you use SPUFI, if one does not already exist. Consider the following rules:

- The name of the data set must conform to standard TSO naming conventions.
- The data set can be empty before you begin the session. You can then add the SQL statements by editing the data set from SPUFI.
- The data set can be either sequential or partitioned, but it must have the following DCB characteristics:
  - A record format (RECFM) of either F or FB.
  - A logical record length (LRECL) of either 79 or 80. Use 80 for any data set that the EXPORT command of QMF did not create.
- Data in the data set can begin in column 1. It can extend to column 71 if the logical record length is 79, and to column 72 if the logical record length is 80. SPUFI assumes that the last 8 bytes of each record are for sequence numbers.

If you use this panel a second time, the name of the data set you previously used displays in the field DATA SET NAME. To create a new member of an existing partitioned data set, change only the member name.

4 OUTPUT DATA SET NAME
Enter the name of a data set to receive the output of the SQL statement. You do not need to allocate the data set before you do this.

If the data set exists, the new output replaces its content. If the data set does not exist, Db2 allocates a data set on the device type specified on the CURRENT SPUFI DEFAULTS panel and then catalogs the new data set. The device must be a direct-access storage device, and you must be authorized to allocate space on that device.

Attributes required for the output data set are:

- Organization: sequential
- Record format: F, FB, FBA, V, VB, or VBA
- Record length: 80 to 32768 bytes, not less than the input data set

"Executing SQL by using SPUFI" on page 939 shows the simplest choice, entering **RESULT**. SPUFI allocates a data set named *userid*.RESULT and sends all output to that data set. If a data set named *userid*.RESULT already exists, SPUFI sends Db2 output to it, replacing all existing data.

5 CHANGE DEFAULTS
Enables you to change control values and characteristics of the output data set and format of your SPUFI session. If you specify **Y**(YES) you can look at the SPUFI defaults panel. See "Changing SPUFI defaults" on page 944 for more information about the values you can specify and how they affect SPUFI processing and output characteristics. You do not need to change the SPUFI defaults for this example.

6 EDIT INPUT
To edit the input data set, leave **Y**(YES) on line 6. You can use the ISPF editor to create a new member of the input data set and enter SQL statements in it. (To process a data set that already contains a set

of SQL statements you want to execute immediately, enter **N** (NO). Specifying N bypasses the step 3 described in "Executing SQL by using SPUFI" on page 939.)

7 EXECUTE
To execute SQL statements contained in the input data set, leave **Y**(YES) on line 7.

SPUFI handles the SQL statements that can be dynamically prepared.

8 AUTOCOMMIT
To make changes to the Db2 data permanent, leave **Y**(YES) on line 8. Specifying **Y** makes SPUFI issue COMMIT if all statements execute successfully. If all statements do not execute successfully, SPUFI issues a ROLLBACK statement, which deletes changes already made to the file (back to the last commit point).

If you specify **N**, Db2 displays the SPUFI COMMIT OR ROLLBACK panel after it executes the SQL in your input data set. That panel prompts you to COMMIT, ROLLBACK, or DEFER any updates made by the SQL. If you enter DEFER, you neither commit nor roll back your changes.

9 BROWSE OUTPUT
To look at the results of your query, leave **Y**(YES) on line 9. SPUFI saves the results in the output data set. You can look at them at any time, until you delete or write over the data set.

10 CONNECT LOCATION
Specify the name of the database server, if applicable, to which you want to submit SQL statements. SPUFI then issues a type 2 CONNECT statement to this server.

SPUFI is a locally bound package. SQL statements in the input data set can process only if the CONNECT statement is successful. If the connect request fails, the output data set contains the resulting SQL return codes and error messages.

Related reference

Characteristics of SQL statements in Db2 for z/OS (Db2 SQL)
COMMIT (Db2 SQL)
ROLLBACK (Db2 SQL)

Changing SPUFI defaults

Before you execute SQL statements in SPUFI, you can change the default execution behavior, such as the SQL terminator and the isolation level.

About this task

SPUFI provides default values the first time that you use SPUFI for all options except the Db2 subsystem name. Any changes that you make to these values remain in effect until you change the values again.

Procedure

To change the SPUFI defaults:

1. On the SPUFI panel, specify YES in the CHANGE DEFAULTS field.
2. Press Enter.
   The CURRENT SPUFI DEFAULTS panel opens. The following figure shows the initial default values.

```
DSNESP02                 CURRENT SPUFI DEFAULTS               SSID: DSN
===>
Enter the following to control your SPUFI session:
 1  SQL TERMINATOR .. ===> ;       (SQL Statement Terminator)
 2  ISOLATION LEVEL   ===> RR      (RR=Repeatable Read, CS=Cursor Stability)
                                    UR=Uncommitted Read)
 3  MAX SELECT LINES  ===> 250     (Maximum lines to be returned from a SELECT)
 4  ALLOW SQL WARNINGS===> NO      (Continue fetching after SQL warning)
 5  CHANGE PLAN NAMES ===> NO      (Change the plan names used by SPUFI)
 6  SQL FORMAT ...... ===> SQL     (SQL, SQLCOMNT, or SQLPL)
Output data set characteristics:
 7  SPACE UNIT ...... ===> TRK     (TRK or CYL)
 8  PRIMARY SPACE ... ===> 5       (Primary space allocation 1-999)
 9  SECONDARY SPACE . ===> 6       (Secondary space allocation 0-999)
10  RECORD LENGTH ... ===> 4092    (LRECL= logical record length)
11  BLOCKSIZE ....... ===> 4096    (Size of one block)
12  RECORD FORMAT.... ===> VB      (RECFM= F, FB, FBA, V, VB, or VB)
13  DEVICE TYPE...... ===> SYSDA   (Must be a DASD unit name)
Output format characteristics:
14  MAX NUMERIC FIELD ===> 33      (Maximum width for numeric field)
15  MAX CHAR FIELD .. ===> 80      (Maximum width for character field)
16  COLUMN HEADING .. ===> NAMES   (NAMES, LABELS, ANY, or BOTH)



PRESS: ENTER to process     END to exit    HELP for more information
```

Figure 68. The SPUFI defaults panel

3. Specify any new values in the fields of this panel. All fields must contain a value.
4. Press Enter.

   SPUFI saves your changes and one of the following panels or data sets open:

   - The CURRENT SPUFI DEFAULTS - PANEL 2 panel. This panel opens if you specified YES in the CHANGE PLAN NAMES field.
   - EDIT panel. This panel opens if you specified YES in the EDIT INPUT field on the SPUFI panel.
   - Output data set. This data set opens if you specified NO in the EDIT INPUT field on the SPUFI panel.
   - SPUFI panel. This panel opens if you specified NO for all of the processing options on the SPUFI panel.

   If you press the END key on the CURRENT SPUFI DEFAULTS panel, the SPUFI panel is displayed, and you lose all the changes that you made on the CURRENT SPUFI DEFAULTS panel.
5. If the CURRENT SPUFI DEFAULTS - PANEL 2 panel opens, specify values for the fields on that panel and press Enter. All fields must contain a value.

   Important: If you specify an invalid or incorrect plan name, SPUFI might experience operational errors or your data might be contaminated.

   SPUFI saves your changes and one of the following panels or data sets open:

   - EDIT panel. This panel opens if you specified YES in the EDIT INPUT field on the SPUFI panel.
   - Output data set. This data set opens if you specified NO in the EDIT INPUT field on the SPUFI panel.
   - SPUFI panel. This panel opens if you specified NO for all of the processing options on the SPUFI panel.

Results

Next, continue with one of the following tasks:

- If you want to add SQL statements to the input data set or edit the SQL statements in the input data set, enter SQL statements in SPUFI.
- Otherwise if the input data set already contains the SQL statements that you want to execute, process SQL statements with SPUFI.

Related reference

CURRENT SPUFI DEFAULTS panel

Use the CURRENT SPUFI DEFAULTS panel to specify SPUFI default values.

CURRENT SPUFI DEFAULTS - PANEL 2 panel
Use the CURRENT SPUFI DEFAULTS - PANEL 2 panel to specify default plan name information.

CURRENT SPUFI DEFAULTS panel

Use the CURRENT SPUFI DEFAULTS panel to specify SPUFI default values.

The following descriptions explain the information on the CURRENT SPUFI DEFAULTS panel.

1 SQL TERMINATOR
Specify the character that you use to end each SQL statement. You can specify any character *except* the characters listed in the following table. A semicolon (;) is the default SQL terminator.

Table 153. Invalid special characters for the SQL terminator

Name	Character	Hexadecimal representation
blank		X'40'
comma	,	X'5E'
double quote	"	X'7F'
left parenthesis	(	X'4D'
right parenthesis	)	X'5D'
single quote	'	X'7D'
underscore	_	X'6D'

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons. For example, suppose you choose the character # as the statement terminator. Then a CREATE TRIGGER statement with embedded semicolons looks like the following statement:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

A CREATE PROCEDURE statement with embedded semicolons looks like the following statement:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
  LANGUAGE SQL
  BEGIN
    DECLARE SQLCODE INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET SCODE = SQLCODE;
    UPDATE TBL1 SET COL1 = PARM1;
  END #
```

Be careful to choose a character for the SQL terminator that is not used within the statement.

You can also set or change the SQL terminator within a SPUFI input data set by using the `--#SET TERMINATOR` statement.

2 ISOLATION LEVEL
Specify the isolation level for your SQL statements.

3 MAX SELECT LINES
The maximum number of rows that a SELECT statement can return. To limit the number of rows retrieved, enter another maximum number greater than 1.

4 ALLOW SQL WARNINGS
Enter YES or NO to indicate whether SPUFI will continue to process an SQL statement after receiving SQL warnings:

YES
If a warning occurs when SPUFI executes an OPEN or FETCH for a SELECT statement, SPUFI continues to process the SELECT statement.

NO
If a warning occurs when SPUFI executes an OPEN or FETCH for a SELECT statement, SPUFI stops processing the SELECT statement. If SQLCODE +802 occurs when SPUFI executes a FETCH for a SELECT statement, SPUFI continues to process the SELECT statement.

You can also specify how SPUFI pre-processes the SQL input by using the `--#SET TOLWARN` statement.

5 CHANGE PLAN NAMES
If you enter YES in this field, you can change plan names on a subsequent SPUFI defaults panel, DSNESP07. Enter YES in this field only if you are certain that you want to change the plan names that are used by SPUFI. Consult with your Db2 system administrator if you are uncertain whether you want to change the plan names. Using an invalid or incorrect plan name might cause SPUFI to experience operational errors or it might cause data contamination.

6 SQL FORMAT
Specify how SPUFI pre-processes the SQL input before passing it to Db2. Select one of the following options:

SQL
This is the preferred mode for SQL statements other than SQL procedural language. When you use this option, which is the default, SPUFI collapses each line of an SQL statement into a single line before passing the statement to Db2. SPUFI also discards all SQL comments.

SQLCOMNT
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, behavior is similar to SQL mode, except that SPUFI does not discard SQL comments. Instead, it automatically terminates each SQL comment with a line feed character (hex 25), unless the comment is already terminated by one or more line formatting characters. Use this option to process SQL procedural language with minimal modification by SPUFI.

SQLPL
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, SPUFI retains SQL comments and terminates each line of an SQL statement with a line feed character (hex 25) before passing the statement to Db2. Lines that end with a split token are not terminated with a line feed character. Use this mode to obtain improved diagnostics and debugging of SQL procedural language.

You can also specify how SPUFI pre-processes the SQL input by using the `--#SET SQLFORMAT` statement.

7 SPACE UNIT
Specify how space for the SPUFI output data set is to be allocated.

TRK
Track

CYL
Cylinder

8 PRIMARY SPACE
Specify how many tracks or cylinders of primary space are to be allocated.

9 SECONDARY SPACE
Specify how many tracks or cylinders of secondary space are to be allocated.

10 RECORD LENGTH
The record length must be at least 80 bytes. The maximum record length depends on the device type you use. The default value allows a 32756-byte record.

Each record can hold a single line of output. If a line is longer than a record, the output is truncated, and SPUFI discards fields that extend beyond the record length.

11 BLOCKSIZE
Follow the normal rules for selecting the block size. For record format F, the block size is equal to the record length. For FB and FBA, choose a block size that is an even multiple of LRECL. For VB and VBA only, the block size must be 4 bytes larger than the block size for FB or FBA.

12 RECORD FORMAT
Specify F, FB, FBA, V, VB, or VBA. FBA and VBA formats insert a printer control character after the number of lines specified in the LINES/PAGE OF LISTING field on the DB2I Defaults panel. The record format default is VB (variable-length blocked).

13 DEVICE TYPE
Specify a standard z/OS name for direct-access storage device types. The default is SYSDA. SYSDA specifies that z/OS is to select an appropriate direct access storage device.

14 MAX NUMERIC FIELD
The maximum width of a numeric value column in your output. Choose a value greater than 0. The default is 33.

15 MAX CHAR FIELD
The maximum width of a character value column in your output. DATETIME and GRAPHIC data strings are externally represented as characters, and SPUFI includes their defaults with the default values for character fields. Choose a value greater than 0. The IBM-supplied default is 250.

16 COLUMN HEADING
You can specify NAMES, LABELS, ANY, or BOTH for column headings.

- NAMES uses column names only.
- LABELS (default) uses column labels. Leave the title blank if no label exists.
- ANY uses existing column labels or column names.
- BOTH creates two title lines, one with names and one with labels.

Column names are the column identifiers that you can use in SQL statements. If an SQL statement has an AS clause for a column, SPUFI displays the contents of the AS clause in the heading, rather than the column name. You define column labels with LABEL statements.

Related concepts

Output from SPUFI
SPUFI formats and displays the output data set using the ISPF Browse program.

Related tasks

Changing SPUFI defaults
Before you execute SQL statements in SPUFI, you can change the default execution behavior, such as the SQL terminator and the isolation level.

Executing SQL by using SPUFI
You can execute SQL statements dynamically in a TSO session by using the SPUFI (SQL processor using file input) facility.

CURRENT SPUFI DEFAULTS - PANEL 2 panel

Use the CURRENT SPUFI DEFAULTS - PANEL 2 panel to specify default plan name information.

This panel opens if you specify YES in the CHANGE PLAN NAMES field of the CURRENT SPUFI DEFAULTS panel.

Figure 69 on page 949 shows the initial default values.

```
DSNESP07                CURRENT SPUFI DEFAULTS - PANEL 2        SSID: DSN
===>
Enter the following to control your SPUFI session:
 1 CS ISOLATION PLAN   ===> DSNESPCS (Name of plan for CS isolation level)
 2 RR ISOLATION PLAN   ===> DSNESPRR (Name of plan for RR isolation level)
 3 UR ISOLATION PLAN   ===> DSNESPUR (Name of plan for UR isolation level)

Indicate warning message status:
 4 BLANK CCSID WARNING ===> YES     (Show warning if terminal CCSID is blank)















PRESS: ENTER to process     END to exit    HELP for more information
```

Figure 69. CURRENT SPUFI DEFAULTS - PANEL 2

The following descriptions explain the information on the CURRENT SPUFI DEFAULTS - PANEL 2 panel.

1 CS ISOLATION PLAN
Specify the name of the plan that SPUFI uses when you specify an isolation level of cursor stability (CS). By default, this name is DSNESPCS.

2 RR ISOLATION PLAN
Specify the name of the plan that SPUFI uses when you specify an isolation level of repeatable read (RR). By default, this name is DSNESPRR.

3 UR ISOLATION PLAN
Specify the name of the plan that SPUFI uses when you specify an isolation level of uncommitted read (UR). By default, this name is DSNESPUR.

4 BLANK CCSID ALERT
Indicate whether to receive message DSNE345I when the terminal CCSID setting is blank. A blank terminal CCSID setting occurs when the terminal code page and character set cannot be queried or if they are not supported by ISPF.

Recommendation: To avoid possible data contamination use the default setting of YES, unless you are specifically directed by your Db2 system administrator to use NO.

Setting the SQL terminator character in a SPUFI input data set

In the SPUFI input data set, you can override the SQL terminator character that is specified on the CURRENT SPUFI DEFAULTS panel. The default SQL terminator is a semicolon (;).

About this task

Overriding the default SQL termination character is useful if you need to use a different SQL terminator character for one particular SQL statement.

To set the SQL terminator character in a SPUFI input data set, specify the text --#SET TERMINATOR *character* before that SQL statement to which you want this character to apply. This text specifies that SPUFI is to interpret *character* as a statement terminator. You can specify any single-byte character **except** the characters that are listed in Table 154 on page 950. Choose a character for the SQL terminator that is not used within the statement. The terminator that you specify overrides a terminator that you specified in option 1 of the CURRENT SPUFI DEFAULTS panel or in a previous --#SET TERMINATOR statement.

Table 154. Invalid special characters for the SQL terminator

Name	Character	Hexadecimal representation
blank		X'40'
comma	,	X'5E'
double quote	"	X'7F'
left parenthesis	(	X'4D'
right parenthesis	)	X'5D'
single quote	'	X'7D'
underscore	_	X'6D'

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons. For example, suppose that you choose the character # as the statement terminator. In this case, a CREATE TRIGGER statement with embedded semicolons looks like this:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

Controlling toleration of warnings in SPUFI

When you use SPUFI, you can specify the action that SPUFI is to take when a warning occurs.

About this task

To control the toleration of warnings, specify one of the following TOLWARN control statements:

--#SET TOLWARN NO
: If a warning occurs when SPUFI executes an OPEN or FETCH for SELECT statement, SPUFI stops processing the SELECT statement. If SQLCODE +802 occurs when SPUFI executes a FETCH for a SELECT statement, SPUFI continues to process the SELECT statement.

--#SET TOLWARN YES
: If a warning occurs when SPUFI executes an OPEN or FETCH for SELECT statement, SPUFI continues to process the SELECT statement.

--#SET TOLWARN QUIET
: The same as YES, except that SPUFI suppresses all SQL warning messages from OPEN or FETCH if the SQLCODE is 0 or greater.

Example

The following example activates and then deactivates toleration of SQL warnings:

```
SELECT * FROM MY.T1;
--#SET TOLWARN YES
SELECT * FROM YOUR.T1;
--#SET TOLWARN NO
```

Output from SPUFI

SPUFI formats and displays the output data set using the ISPF Browse program.

An output data set contains the following items for each SQL statement that Db2 executes:

- The executed SQL statement, copied from the input data set
- The results of executing the SQL statement.
- The SQLCODE, and if the execution is unsuccessful, the formatted SQLCA.

At the end of the data set are summary statistics that describe the processing of the input data set as a whole.

For SELECT statements that are executed with SPUFI, the message "`SQLCODE IS 100`" indicates an error-free result. If the message `SQLCODE IS 100` is the only result, Db2 is unable to find any rows that satisfy the condition that is specified in the statement.

For all other types of SQL statements that are executed with SPUFI, the message "`SQLCODE IS 0`" indicates an error-free result.

Example SPUFI output

For example, the sample program returns the following output.

```
 BROWSE-- userid.RESULT                       COLUMNS 001 072
 COMMAND INPUT ===>                                          SCROLL ===> PAGE
 --------+---------+---------+---------+---------+---------+---------+---------+
 SELECT LASTNAME, FIRSTNME, PHONENO                                     00010000
  FROM DSN8B10.EMP                                                      00020000
  WHERE WORKDEPT = 'D11'                                                00030000
  ORDER BY LASTNAME;                                                    00040000
---------+---------+---------+---------+---------+---------+---------+---------+
LASTNAME         FIRSTNME      PHONENO
ADAMSON          BRUCE         4510
BROWN            DAVID         4501
JOHN             REBA          0672
JONES            WILLIAM       0942
LUTZ             JENNIFER      0672
PIANKA           ELIZABETH     3782
SCOUTTEN         MARILYN       1682
STERN            IRVING        6423
WALKER           JAMES         2986
YAMAMOTO         KIYOSHI       2890
YOSHIMURA        MASATOSHI     2890
DSNE610I NUMBER OF ROWS DISPLAYED IS 11
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+----
---------+---------+---------+---------+---------+---------+----
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+----
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 4
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 30
```

Figure 70. Result data set from the sample problem

Formatting rules for SELECT statement results in SPUFI

The results of SELECT statements follow these rules:

- If numeric or character data of a column cannot be displayed completely:
  - Character values and binary values that are too wide truncate on the right.
  - Numeric values that are too wide display as asterisks (*).
  - For columns other than LOB and XML columns, if truncation occurs, the output data set contains a warning message. Because LOB and XML columns are generally longer than the value you choose for field MAX CHAR FIELD on panel CURRENT SPUFI DEFAULTS, SPUFI displays no warning message when it truncates LOB or XML column output.

  You can change the amount of data that is displayed for numeric and character columns by changing values on the CURRENT SPUFI DEFAULTS panel, as described in "Changing SPUFI defaults" on page 944.

- A null value is displayed as a series of hyphens (-).
- A ROWID, BLOB, BINARY, or VARBINARY column value is displayed in hexadecimal.
- A CLOB column value is displayed in the same way as a VARCHAR column value.
- A DBCLOB column value is displayed in the same way as a VARGRAPHIC column value.
- An XML column is displayed in the same way as a LOB column.
- A heading identifies each selected column, and is repeated at the top of each output page. The contents of the heading depend on the value that you specified in the COLUMN HEADING field of the CURRENT SPUFI DEFAULTS panel.

Content of the messages from SPUFI

Messages in the SPUFI output contain the following information:

- The SQLCODE, and if the execution is unsuccessful, the formatted SQLCA. If multiple SQL conditions occur, SPUFI returns the error SQLCODE first, followed by any previous SQLCODE conditions that occurred as the input SQL statement was executed. If the final SQLCODE is an error, most prior SQLCODE warnings can be ignored.
- What character positions of the input data set that SPUFI scanned to find SQL statements. This information helps you check the assumptions that SPUFI made about the location of line numbers (if any) in your input data set.
- Some overall statistics:
  - Number of SQL statements that are processed
  - Number of input records that are read (from the input data set)
  - Number of output records that are written (to the output data set).

Other messages that you could receive from the processing of SQL statements include:

- The number of rows that Db2 processed, that either:
  - Your select operation retrieved
  - Your update operation modified
  - Your insert operation added to a table
  - Your delete operation deleted from a table
- Which columns display truncated data because the data was too wide

Testing an external user-defined function

Some commonly used debugging tools, such as TSO TEST, are not available in the environment where user-defined functions run. You need to use alternative testing strategies.

Testing a user-defined function by using the Debug Tool for z/OS

You can use the Debug Tool for z/OS to test Db2 for z/OS user-defined functions that are written in any of the supported languages. The Debug Tool for z/OS works with Language Environment.

About this task

You can use the Debug Tool either interactively or in batch mode. To test your user-defined function using the Debug Tool, you must have the Debug Tool installed on the z/OS system where the user-defined function runs.

Procedure

To test a user-defined function by using the Debug Tool for z/OS, choose one of the following approaches:

- To use the Debug Tool interactively:

a) Compile the user-defined function with the TEST option.

This places information in the program that the Debug Tool uses.

b) Invoke the Debug Tool. One way to do that is to specify the Language Environment run time TEST option. The TEST option controls when and how the Debug Tool is invoked. The most convenient place to specify run time options is with the RUN OPTIONS clause of CREATE FUNCTION or ALTER FUNCTION.
For example, suppose that you code this option:

```
TEST(ALL,*,PROMPT,JBJONES%SESSNA:)
```

The parameter values cause the following things to happen:

ALL
The Debug Tool gains control when an attention interrupt, abend, or program or Language Environment condition of Severity 1 and above occurs.

*****
Debug commands will be entered from the terminal.

PROMPT
The Debug Tool is invoked immediately after Language Environment initialization.

JBJONES%SESSNA:
The Debug Tool initiates a session on a workstation identified to APPC as JBJONES with a session ID of SESSNA.

c) If you want to save the output from your debugging session, issue a command that names a log file. For example, the following command starts logging to a file on the workstation called dbgtool.log.

```
SET LOG ON FILE dbgtool.log;
```

This should be the first command that you enter from the terminal or include in your commands file.

- To use the Debug Tool in batch mode:

a) If you plan to use the Language Environment run time TEST option to invoke the Debug Tool, compile the user-defined function with the TEST option.

This places information in the program that the Debug Tool uses during a debugging session.

b) Allocate a log data set to receive the output from the Debug Tool. Put a DD statement for the log data set in the startup procedure for the stored procedures address space.

c) Enter commands in a data set that you want the Debug Tool to execute. Put a DD statement for that data set in the startup procedure for the stored procedures address space. To define the data set that contains the commands to the Debug Tool, specify its data set name or DD name in the TEST run time option.
For example, this option tells the Debug Tool to look for the commands in the data set that is associated with DD name TESTDD:

```
TEST(ALL,TESTDD,PROMPT,*)
```

The first command in the commands data set should be:

```
SET LOG ON FILE ddname;
```

This command directs output from your debugging session to the log data set you defined in step 2. For example, if you defined a log data set with DD name INSPLOG in the start-up procedure for the stored procedures address space, the first command should be:

```
SET LOG ON FILE INSPLOG;
```

d) Invoke the Debug Tool.

The following are two possible methods for invoking the Debug Tool:

- Specify the Language Environment run time TEST option. The most convenient place to do that is in the RUN OPTIONS parameter of CREATE FUNCTION or ALTER FUNCTION.
- Put CEETEST calls in the user-defined function source code. If you use this approach for an existing user-defined function, you must compile, link-edit, and bind the user-defined function again. Then you must issue the STOP FUNCTION SPECIFIC and START FUNCTION SPECIFIC commands to reload the user-defined function.

  You can combine the Language Environment run time TEST option with CEETEST calls. For example, you might want to use TEST to name the commands data set but use CEETEST calls to control when the Debug Tool takes control.

You can combine the Language Environment run time TEST option with CEETEST calls. For example, you might want to use TEST to name the commands data set but use CEETEST calls to control when the Debug Tool takes control.

Related reference

CREATE FUNCTION (Db2 SQL)

Debug Tool for z/OS

Testing a user-defined function by routing the debugging messages to SYSPRINT

You can include simple print statements in your user-defined function code that you route to SYSPRINT. Then use System Display and Search Facility (SDSF) to examine the SYSPRINT contents while the WLM-established stored procedure address space is running.

About this task

You can serialize I/O by running the WLM-established stored procedure address space with NUMTCB=1.

Testing a user-defined function by using driver applications

You can write a small driver application that calls a user-defined function as a subprogram and passes the parameter list for the user-defined function. You can then test and debug the user-defined function as a normal Db2 application under TSO.

About this task

You can then use TSO TEST and other commonly used debugging tools.

Testing a user-defined function by using SQL INSERT statements

You can use SQL to insert debugging information into a Db2 table. This allows other machines in the network (such as workstations) to easily access the data in the table by using DRDA access.

About this task

Db2 discards the debugging information if the application executes the ROLLBACK statement. To prevent the loss of the debugging data, code the calling application so that it retrieves the diagnostic data before executing the ROLLBACK statement.

Debugging stored procedures

When debugging stored procedures, you might need to use different techniques than you would use for regular application programs. For example, some commonly used debugging tools, such as TSO TEST, are not available in the environment where stored procedures run.

Procedure

To debug a stored procedure, perform one or more of the following actions:

- Take one or more of the following general actions, which are appropriate in many situations with stored procedures:
  - Ensure that all stored procedures are written to handle any SQL errors.
  - Debug stored procedures as stand-alone programs on a workstation.

    If you have debugging tools on a workstation, consider doing most of your development and testing on a workstation before installing a stored procedure on z/OS. This technique results in very little debugging activity on z/OS.
  - Record stored procedure debugging messages to a disk file or JES spool file.
  - Store debugging information in a table. This technique is especially useful for remote stored procedures.
  - For native SQL procedures, call the sample debug stored procedures to write trace messages to a buffer.
  - Use the DISPLAY command to view information about particular stored procedures, including statistics and thread information.
  - In the stored procedure that you are debugging, issue DISPLAY commands. You can view the DISPLAY results in the SDSF output. The DISPLAY results can help you find information about the started task that is associated with the address space for the WLM application environment.
  - If necessary, use the STOP PROCEDURE command to stop calls to one or more problematic stored procedures. You can restart them later.
- If your stored procedures address space has the CEEDUMP data set allocated, look at the diagnostic information in the CEEDUMP output.
- For COBOL, C, and C++ stored procedures, use the Debug Tool for z/OS.
- For COBOL stored procedures, compile the stored procedure with the option TEST(SYM) if you want a formatted local variable dump to be included in the CEEDUMP output.
- For native SQL procedures, external SQL procedures, and Java stored procedures, use the Unified Debugger.
- For external stored procedures, consider taking one or both of the following actions:
  - Use a driver application.
  - Create or alter the stored procedure definition to include the PARAMETER STYLE SQL option. This option enables the stored procedure to share any error information with the calling application. Ensure that your procedure follows linkage conventions for stored procedures.
- If you changed a stored procedure or a startup JCL procedure for a WLM application environment, determine whether you need to refresh the WLM environment. You must refresh the WLM environment before certain stored procedure changes take effect.

Related tasks

Handling SQL conditions in an SQL procedure
In an SQL procedure, you can specify how the program should handle certain SQL errors and SQL warnings.

Displaying information about stored procedures with Db2 commands (Db2 Administration Guide)
Refreshing WLM application environments for stored procedures (Db2 Administration Guide)
Implementing Db2 stored procedures (Db2 Administration Guide)

Related reference

Linkage conventions for external stored procedures
The linkage convention for a stored procedure can be either GENERAL, GENERAL WITH NULLS, or SQL. These linkage conventions apply to only external stored procedures.

-START PROCEDURE (Db2) (Db2 Commands)

-STOP PROCEDURE (Db2) (Db2 Commands)

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Debugging stored procedures with the Debug Tool and IBM VisualAge COBOL

If you have VisualAge® COBOL installed on your workstation and the Debug Tool installed on your z/OS system, you can use the VisualAge COBOL Edit/Compile/Debug component with the Debug Tool to debug COBOL stored procedures that run in a WLM-established stored procedures address space.

About this task

Before you begin debugging, write your COBOL stored procedure and set up the WLM environment.

Procedure

To debug with the Debug Tool and IBM VisualAge COBOL:

1. When you compile the stored procedure, specify the TEST and SOURCE options. Ensure that the source listing is stored in a permanent data set. VisualAge COBOL displays the source listing during the debug session.
2. When you define the stored procedure, include run time option TEST with the suboption VADTCPIP&*ipaddr* in your RUN OPTIONS argument.

   VADTCPIP& tells the Debug Tool that it is interfacing with a workstation that runs VisualAge COBOL and is configured for TCP/IP communication with your z/OS system. *ipaddr* is the IP address of the workstation on which you display your debug information. For example, the RUN OPTIONS value in the following stored procedure definition indicates that debug information should go to the workstation with IP address 9.63.51.17:

```
CREATE PROCEDURE WLMCOB
 (IN INTEGER, INOUT VARCHAR(3000), INOUT INTEGER)
 MODIFIES SQL DATA
 LANGUAGE COBOL EXTERNAL
 PROGRAM TYPE MAIN
 WLM ENVIRONMENT WLMENV1
 RUN OPTIONS 'POSIX(ON),TEST(,,,VADTCPIP&9.63.51.17:*)'
```

3. In the JCL startup procedure for WLM-established stored procedures address space, add the data set name of the Debug Tool load library to the STEPLIB concatenation. For example, suppose that ENV1PROC is the JCL procedure for application environment WLMENV1. The modified JCL for ENV1PROC might look like this:

```
//DSNWLM   PROC RGN=0K,APPLENV=WLMENV1,DB2SSN=DSN,NUMTCB=8
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//        PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB  DD  DISP=SHR,DSN=DSN1110.RUNLIB.LOAD
//         DD  DISP=SHR,DSN=CEE.SCEERUN
//         DD  DISP=SHR,DSN=DSN1110.SDSNLOAD
//         DD  DISP=SHR,DSN=EQAW.SEQAMOD <== DEBUG TOOL
```

4. On the workstation, start the VisualAge Remote Debugger daemon.

   This daemon waits for incoming requests from TCP/IP.
5. Call the stored procedure.

   When the stored procedure starts, a window that contains the debug session is displayed on the workstation. You can then execute Debug Tool commands to debug the stored procedure.

Related reference

Debug Tool for z/OS

Debugging a C language stored procedure with the Debug Tool and C/C++ Productivity Tools for z/OS

You can debug a C or C++ stored procedure that runs in a WLM-established stored procedures address space. You must have the C/C++ Productivity Tools for z/OS installed on your workstation and the Debug Tool installed on your z/OS system.

About this task

The code against which you run the debug tools is the C source program that is produced by the program preparation process for the stored procedure.

Before you begin debugging, write your C++ stored procedure and set up the WLM environment.

Procedure

To test the stored procedure with the Distributed Debugger feature of the C/C++ Productivity Tools for z/OS and the Debug Tool:

1. When you define the stored procedure, include run time option TEST with the suboption VADTCPIP&*ipaddr* in your RUN OPTIONS argument.

   VADTCPIP& tells the Debug Tool that it is interfacing with a workstation that runs VisualAge C++ and is configured for TCP/IP communication with your z/OS system. *ipaddr* is the IP address of the workstation on which you display your debug information. For example, this RUN OPTIONS value in a stored procedure definition indicates that debug information should go to the workstation with IP address 9.63.51.17:

   ```
   RUN OPTIONS 'POSIX(ON),TEST(,,,VADTCPIP&9.63.51.17:*)'
   ```

2. Precompile the stored procedure. Ensure that the modified source program that is the output from the precompile step is in a permanent, catalogued data set.
3. Compile the output from the precompile step. Specify the TEST, SOURCE, and OPT(0) compiler options.
4. In the JCL startup procedure for the stored procedures address space, add the data set name of the Debug Tool load library to the STEPLIB concatenation. For example, suppose that ENV1PROC is the JCL procedure for application environment WLMENV1. The modified JCL for ENV1PROC might look like this:

   ```
   //DSNWLM   PROC RGN=0K,APPLENV=WLMENV1,DB2SSN=DSN,NUMTCB=8
   //IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
   //        PARM='&DB2SSN,&NUMTCB,&APPLENV'
   //STEPLIB  DD  DISP=SHR,DSN=DSN1110.RUNLIB.LOAD
   //         DD  DISP=SHR,DSN=CEE.SCEERUN
   //         DD  DISP=SHR,DSN=DSN1110.SDSNLOAD
   //         DD  DISP=SHR,DSN=EQAW.SEQAMOD <== DEBUG TOOL
   ```

5. On the workstation, start the Distributed Debugger daemon.

   This daemon waits for incoming requests from TCP/IP.
6. Call the stored procedure.

   When the stored procedure starts, a window that contains the debug session is displayed on the workstation. You can then execute Debug Tool commands to debug the stored procedure.

Related reference

Debug Tool for z/OS

Debugging stored procedures by using the Unified Debugger

You can use the Unified Debugger to remotely debug native SQL procedures, external SQL procedures, and Java stored procedures that execute on Db2 for z/OS servers. The Unified Debugger also supports debugging nested stored procedure calls.

About this task

With the Unified Debugger, you can observe the execution of the procedure code, set breakpoints for lines, and view or modify variable values.

Procedure

To debug stored procedures by using the Unified Debugger:

1. Set up the Unified Debugger by performing the following steps:
   a) Ensure that job DSNTIJRT successfully created the stored procedures that provide server support for the Unified Debugger. This job is run during the installation and migration process. The stored procedures that this job creates must run in WLM environments.

      Recommendation: Initially, define and use the Db2 core WLM environment DSNWLM_GENERAL to run the SYSPROC.DBG_RUNSESSIONMANAGER stored procedure and core WLM environment DSNWLM_DEBUGGER to run the other stored procedures for Unified debugger.
   b) Define the debug mode characteristics for the stored procedure that you want to debug by completing one of the following actions:
      - For a native SQL procedure, define the procedure with the ALLOW DEBUG MODE option and the WLM ENVIRONMENT FOR DEBUG MODE option. If the procedure already exists, you can use the ALTER PROCEDURE statement to specify these options.
      - For an external SQL procedure, use DSNTPSMP or IBM Data Studio to build the SQL procedure with the BUILD_DEBUG option.
      - For a Java stored procedure, define the procedure with the ALLOW DEBUG MODE option, select an appropriate WLM environment for Java debugging, and compile the Java code with the -G option.
   c) Grant the DEBUGSESSION privilege to the user who runs the debug client.
2. Include breakpoints in your routines or executable files.
3. Follow the instructions for debugging stored procedures in the information for IBM Data Studio.

Related concepts

Java stored procedures and user-defined functions (Db2 Application Programming for Java)

Related tasks

Creating an external SQL procedure by using DSNTPSMP
The SQL procedure processor, DSNTPSMP, is one of several methods that you can use to create and prepare an external SQL procedure. DSNTPSMP is a REXX stored procedure that you can invoke from your application program.

Developing database routines (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio)

Related reference

Sample programs to help you prepare and run external SQL procedures
Db2 provides sample jobs to help you prepare and run external SQL procedures. All samples are in data set DSN1110.SDSNSAMP. Before you can run the samples, you must customize them for your installation.

ALTER PROCEDURE (SQL - native) (Db2 SQL)

CREATE PROCEDURE (SQL - native) (Db2 SQL)

Related information

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)

Tracing native SQL procedures

You can debug native SQL procedures by calling a set of native SQL procedures that let you display trace messages.

Procedure

To debug native SQL procedures by writing trace messages:

1. Test whether the message trace is enabled by checking the value of the DSN8.DBMS_ENABLE global variable.
2. If the message trace is not enabled, enable the trace buffer by calling the DSN8.ENABLE procedure.
3. Write information to the message buffer by calling the DSN8.PUT_LINE or DSN8.PUT procedure. DSN8.PUT_LINE writes a line with an end-of-line sequence. DSN8.PUT writes a line without an end-of-line sequence.

   You can improve readability of the data in the message buffer by calling the DSN8.NEW_LINE procedure to add end-of-line sequences to the message buffer.
4. Retrieve information from the message buffer. You can retrieve a single line by calling the DSN8.GET_LINE procedure. You can retrieve multiple lines by creating an array variable of type DSN8.GRPHICARR, and then calling the DSN8.GET_LINES procedure to retrieve the data into the array variable.
5. Disable the trace output buffer by calling the DSN8.DISABLE procedure.

Example

This example demonstrates how an SQL procedure can write messages to a trace buffer, read the messages from the trace buffer, and insert the messages into a table.

Before you call this procedure, you need to create a table like this:

```
CREATE TABLE TRACE_CONTENT
 (ROW INTEGER,
 CONTENT VARGRAPHIC(100))
 CCSID UNICODE
```

Procedure PUT_GET demonstrates the trace procedures that you can include in your native SQL procedures.

```
CREATE PROCEDURE PUT_GET()
BEGIN
 DECLARE LINE_ARRAY DSN8.GRPHICARR; -- GLOBAL VARIABLE ARRAY
                                    -- FOR GET_LINES
 DECLARE I INT;                     -- COUNTER FOR GET_LINES
 -- IF TRACING IS NOT ALREADY ENABLED, ENABLE IT.
 IF DSN8.DBMS_ENABLE = 'Y' THEN
  CALL DSN8.PUT_LINE('Trace is already enabled');
 ELSE
  -- ENABLE TRACING WITH A 100-BYTE
BUFFER.

  CALL DSN8.ENABLE(100);
 END IF;
 -- PUT A STRING IN THE BUFFER WITHOUT AN
 -- END-OF-LINE CHARACTER SEQUENCE.
 CALL DSN8.PUT ('String with no end-of-line sequence ');
 -- PUT A STRING IN THE BUFFER WITH AN
 -- END-OF-LINE CHARACTER SEQUENCE.
 CALL DSN8.PUT_LINE ('String with end-of-line sequence');
 -- PUT A STRING IN THE BUFFER WITHOUT AN
 -- END-OF-LINE CHARACTER SEQUENCE, AND FOLLOW IT WITH
 -- A NEW_LINE CALL TO PUT IN AN END-OF-LINE SEQUENCE.
 CALL DSN8.PUT('Strings with no end-of-line sequence, ');
 CALL DSN8.PUT('followed by new-line sequence');
 CALL DSN8.NEW_LINE();
 SET I = 3;
 -- GET AT MOST THREE LINES FROM THE TRACE
BUFFER.
 CALL DSN8.GET_LINES(LINE_ARRAY,I);
```

```
 -- INSERT THE LINES FROM THE TRACE BUFFER INTO A TABLE.
 INSERT INTO TRACE_CONTENT VALUES(1, LINE_ARRAY??(1??));
 INSERT INTO TRACE_CONTENT VALUES(2, LINE_ARRAY??(2??));
 --DISABLE THE TRACE.
 CALL DSN8.DISABLE  ;
END#
```

To see the trace messages, issue the CALL PUT_GET statement, and then issue a SELECT statement like this one:

```
SELECT ROW, CONTENT FROM TRACE_CONTENT
```

The results look like these:

ROW	CONTENT
1	String with no end-of-line sequence String with end-of-line sequence
2	Strings with no end-of-line sequence, followed by end-of-line sequence

Related reference

DSN8.ENABLE stored procedure (Db2 SQL)
DSN8.DISABLE stored procedure (Db2 SQL)
DSN8.NEW_LINE stored procedure (Db2 SQL)
DSN8.GET_LINE stored procedure (Db2 SQL)
DSN8.GET_LINES stored procedure (Db2 SQL)
DSN8.PUT stored procedure (Db2 SQL)
DSN8.PUT_LINE stored procedure (Db2 SQL)
Objects that are used by the sample trace stored procedures (Db2 SQL)

Debugging stored procedures with the Debug Tool for z/OS

You can use the Debug Tool to test z/OS stored procedures that are written in any of the compiled languages that the Debug Tool supports. You can test these stored procedures either interactively or in batch mode.

About this task

Using Debug Tool interactively: To test a stored procedure interactively using the Debug Tool, you must have the Debug Tool installed on the z/OS system where the stored procedure runs.

Procedure

To debug your stored procedure using the Debug Tool:

1. Compile the stored procedure with option TEST. This places information in the program that the Debug Tool uses during a debugging session.
2. Invoke the Debug Tool. One way to do that is to specify the Language Environment run time option TEST. The TEST option controls when and how the Debug Tool is invoked. The most convenient place to specify run time options is in the RUN OPTIONS parameter of the CREATE PROCEDURE or ALTER PROCEDURE statement for the stored procedure.

   For example, you can code the TEST option using the following parameters:

   ```
   TEST(ALL,*,PROMPT,JBJONES%SESSNA:)
   ```

   The following table lists the effects that each parameter has on the Debug Tool:

Table 155. Effects of the TEST option parameters on the Debug Tool

Parameter value	Effect on the Debug Tool
ALL	The Debug Tool gains control when an attention interrupt, ABEND, or program or Language Environment condition of Severity 1 and above occurs. Debug commands will be entered from the terminal.
PROMPT	The Debug Tool is invoked immediately after Language Environment initialization.
JBJONES%SESSNA:	The Debug Tool initiates a session on a workstation identified to APPC/MVS as JBJONES with a session ID of SESSNA.

3. If you want to save the output from your debugging session, issue the following command:

```
SET LOG ON FILE dbgtool.log;
```

This command saves a log of your debugging session to a file on the workstation called dbgtool.log. This should be the first command that you enter from the terminal or include in your commands file.

Results

Using Debug Tool in batch mode: To test your stored procedure in batch mode, you must have the Debug Tool installed on the z/OS system where the stored procedure runs. To debug your stored procedure in batch mode using the Debug Tool, complete the following steps:

- Compile the stored procedure with option TEST, if you plan to use the Language Environment run time option TEST to invoke the Debug Tool. This places information in the program that the Debug Tool uses during a debugging session.
- Allocate a log data set to receive the output from the Debug Tool. Put a DD statement for the log data set in the start-up procedure for the stored procedures address space.
- Enter commands in a data set that you want the Debug Tool to execute. Put a DD statement for that data set in the start-up procedure for the stored procedures address space. To define the commands data set to the Debug Tool, specify the commands data set name or DD name in the TEST run time option. For example, to specify that the Debug Tool use the commands that are in the data set that is associated with the DD name TESTDD, include the following parameter in the TEST option:

  ```
  TEST(ALL,TESTDD,PROMPT,*)
  ```

  The first command in the commands data set should be:

  ```
  SET LOG ON FILE ddname;
  ```

  This command directs output from your debugging session to the log data set that you defined in the previous step. For example, if you defined a log data set with DD name INSPLOG in the stored procedures address space start-up procedure, the first command should be the following command:

  ```
  SET LOG ON FILE INSPLOG;
  ```

- Invoke the Debug Tool. The following are two possible methods for invoking the Debug Tool:
  - Specify the run time option TEST. The most convenient place to do that is in the RUN OPTIONS parameter of the CREATE PROCEDURE or ALTER PROCEDURE statement for the stored procedure.

- Put CEETEST calls in the stored procedure source code. If you use this approach for an existing stored procedure, you must recompile, re-link, and bind it, and issue the STOP PROCEDURE and START PROCEDURE commands to reload the stored procedure.

  You can combine the run time option TEST with CEETEST calls. For example, you might want to use TEST to name the commands data set but use CEETEST calls to control when the Debug Tool takes control.

Related reference

Debug Tool for z/OS

Recording stored procedure debugging messages in a file

You can debug external stored procedures and external SQL procedures by recording debugging messages in a disk file or in a JES spool file. You cannot use this debugging technique for native SQL procedures or Java stored procedures.

Procedure

To record stored procedure debugging messages in a file:

1. Specify the Language Environment (LE) MSGFILE run time option for the stored procedure. This option identifies where LE is to write the debugging messages. To specify this option, include the RUN OPTIONS clause in either the CREATE PROCEDURE statement or an ALTER PROCEDURE statement.

   Specify the following MSGFILE parameters:

   - Use the first MSGFILE parameter to specify the JCL DD statement that identifies the data set for the debugging messages. You can direct debugging messages to a disk file or JES spool file. To prevent multiple procedures from sharing a data set, ensure that you specify a unique DD statement.
   - Use the ENQ option to serialize I/O to the message file. This action is necessary, because multiple TCBs can be active in the stored procedure address space. Alternatively, if you debug your applications infrequently or on a Db2 test system, you can serialize I/O by temporarily running the stored procedures address space with NUMTCB=1 in the stored procedures address space start-up procedure.
2. For each instance of MSGFILE that you specify, add a DD statement to the JCL procedure that is used to start the stored procedures address space.

Related reference

ALTER PROCEDURE (external) (Db2 SQL)

ALTER PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

CREATE PROCEDURE (external) (Db2 SQL)

CREATE PROCEDURE (SQL - external) (deprecated) (Db2 SQL)

GRANT (system privileges) (Db2 SQL)

Using Language Environment MSGFILE (z/OS Language Environment Programming Guide)

Driver applications for debugging procedures

You can write a small driver application that calls the stored procedure as a subprogram and passes the parameter list that the stored procedure supports. You can then test and debug the stored procedure as a normal Db2 application under TSO.

Using this method, you can use TSO TEST and other commonly used debugging tools.

Restriction: You cannot use this technique for SQL procedures

Db2 tables that contain debugging information

You can use SQL statements to insert debugging information into a Db2 table. Inserting this information into a table enables other machines in the network (such as a workstation) to easily access the data in the table by using DRDA access.

Db2 discards the debugging information if the application executes the ROLLBACK statement. To prevent the loss of the debugging data, code the calling application so that it retrieves the diagnostic data before executing the ROLLBACK statement.

Debugging an application program

Many sites have guidelines regarding what to do if a program abnormally terminates.

About this task

For information about the compiler or assembler test facilities, see the publications for the compiler or CODE/370. The compiler publications include information about the appropriate debugger for the language you are using.

You can also use ISPF Dialog Test to debug your program. You can run all or portions of your application, examine the results, make changes, and rerun it.

Related reference

Dialog test (option 7) (z/OS ISPF User's Guide Vol II)

Locating the problem in an application

If your program does not run correctly, you need to isolate the problem. You should check several items.

About this task

Those items are:

- Output from the precompiler, which consists of errors and warnings. Ensure that you have resolved all errors and warnings.
- Output from the compiler or assembler. Ensure that you have resolved all error messages.
- Output from the linkage editor.
  - Have you resolved all external references?
  - Have you included all necessary modules in the correct order?
  - Did you include the correct language interface module? The correct language interface module is:
    - DSNELI or DSNULI for TSO
    - DFSLI000 for IMS
    - DSNCLI or DSNULI for CICS
    - DSNALI or DSNULI for the call attachment facility
    - DSNRLI or DSNULI for the Resource Recovery Services attachment facility
  - Did you specify the correct entry point to your program?
- Output from the bind process.
  - Have you resolved all error messages?
  - Did you specify a plan name? If not, the bind process assumes that you want to process the DBRM for diagnostic purposes, but that you do not want to produce an application plan.
  - Have you specified all the packages that are associated with the programs that make up the application and their partitioned data set (PDS) names in a single application plan?
- Your JCL.

IMS

- If you are using IMS, have you included the DL/I option statement in the correct format?
- Have you included the region size parameter in the EXEC statement? Does it specify a region size that is large enough for the required storage for the Db2 interface, the TSO, IMS, or CICS system, and your program?
- Have you included the names of all data sets (Db2 and non-Db2) that the program requires?

- Your program.

  You can also use dumps to help localize problems in your program. For example, one of the more common error situations occurs when your program is running and you receive a message that it abended. In this situation, your test procedure might be to capture a TSO dump. To do so, you must allocate a SYSUDUMP or SYSABEND dump data set before calling Db2. When you press the ENTER key (after the error message and READY message), the system requests a dump. You then need to use the FREE command to deallocate the dump data set.

Error and warning messages from the precompiler

In some circumstances, the statements that the Db2 precompiler generates might produce compiler or assembly error messages. You need to know why the messages occur when you compile Db2-produced source statements.

SYSTERM output from the precompiler

The SYSTERM output provides a brief summary of the results from the precompiler, all error messages that the precompiler generated, and the statement that is in error, when possible.

The Db2 precompiler provides SYSTERM output when you allocate the DD name SYSTERM. If you use the program preparation panels to prepare and run your program, DB2I allocates SYSTERM according to the TERM option that you specify.

You can use the line number that is provided in each error message in the SYSTERM output to locate the failing source statement.

Figure 71 on page 964 shows the format of SYSTERM output.

```
DB2 SQL PRECOMPILER         MESSAGES

DSNH104I E     DSNHPARS LINE 32 COL 26  ILLEGAL SYMBOL "X"  VALID SYMBOLS ARE:, FROM[1]
SELECT VALUE INTO HIPPO X;[2]

DB2 SQL PRECOMPILER         STATISTICS
SOURCE STATISTICS[3]
  SOURCE LINES READ: 36
  NUMBER OF SYMBOLS: 15
  SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 1848
THERE WERE 1 MESSAGES FOR THIS PROGRAM.[4]
THERE WERE 0 MESSAGES SUPPRESSED BY THE FLAG OPTION.[5]
111664 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.[6]
RETURN CODE IS 8[7]
```

Figure 71. Db2 precompiler SYSTERM output

Notes:

1. Error message.
2. Source SQL statement.
3. Summary statements of source statistics.
4. Summary statement of the number of errors that were detected.
5. Summary statement that indicates the number of errors that were detected but not printed. This situation might occur if you specify a FLAG option other than I.

6. Storage requirement statement that indicates how many bytes of working storage that the Db2 precompiler actually used to process your source statements. That value helps you determine the storage allocation requirements for your program.
7. Return code: 0 = success, 4 = warning, 8 = error, 12 = severe error, and 16 = unrecoverable error.

SYSPRINT output from the precompiler

SYSPRINT output from the Db2 precompiler shows the results of the precompile operation. This output can also include a list of the options that were used, a source code listing, and a host variable cross-reference listing.

When you use the program preparation panels to prepare and run your program, Db2 allocates SYSPRINT according to TERM option that you specify (on line 12 of the PROGRAM PREPARATION: COMPILE, PRELINK, LINK, AND RUN panel). As an alternative, when you use the DSNH command procedure (CLIST), you can specify PRINT(TERM) to obtain SYSPRINT output at your terminal, or you can specify PRINT(*qualifier*) to place the SYSPRINT output into a data set named *authorizationID*.*qualifier*.PCLIST. Assuming that you do not specify PRINT as LEAVE, NONE, or TERM, Db2 issues a message when the precompiler finishes, telling you where to find your precompiler listings. This helps you locate your diagnostics quickly and easily.

The SYSPRINT output can provide information about your precompiled source module if you specify the options SOURCE and XREF when you start the Db2 precompiler.

The format of SYSPRINT output is as follows:

- A list of the Db2 precompiler options that are in effect during the precompilation (if you did not specify NOOPTIONS).
- A list of your source statements (only if you specified the SOURCE option). An example is shown in Figure 72 on page 966.
- A list of the symbolic names used in SQL statements (this listing appears only if you specify the XREF option). An example is show in Figure 73 on page 966.
- A summary of the errors that are detected by the Db2 precompiler and a list of the error messages that are generated by the precompiler. An example is shown in

The following code shows an example list of Db2 precompiler options as it is displayed in the SYSPRINT output.

```
DB2 SQL PRECOMPILER VERSION 11 REL. 1.0

OPTIONS SPECIFIED: HOST(PLI),SOURCE,XREF,STDSQL(NO),TWOPASS
DSNHDECP LOADED FROM - (USER99.RELM.TESTLIB(DSNHDECP))
OPTIONS USED - SPECIFIED OR DEFAULTED
APOST
APOSTSQL
ATTACH(TSO)
CCSID(37)
CONNECT(2)
DEC(15)
FLAG(I)
FLOAT(S390)
HOST(PLI)
LINECOUNT(60)
MARGINS(2,72)
NEWFUN(V11)
OPTIONS
PERIOD
SOURCE
SQL(DB2)
STDSQL(NO)
TWOPASS
XREF
```

Notes:

1. This section lists the options that are specified at precompilation time. This list does not appear if one of the precompiler option is NOOPTIONS.

2. This section lists the options that are in effect, including defaults, forced values, and options that you specified. The Db2 precompiler overrides or ignores any options that you specify that are inappropriate for the host language.

The following figure shows an example list of source statements as it is displayed in the SYSPRINT output.

```
DB2 SQL PRECOMPILER       TMN5P40:PROCEDURE OPTIONS (MAIN):           PAGE 2

     1        TMN5P40:PROCEDURE OPTIONS(MAIN) ;                       00000100
     2        /*******************************************************00000200
     3        *      program description and prologue                       00000300
    ⋮
  1324        /*************************************************/     00132400
  1325        /*  GET INFORMATION ABOUT THE PROJECT FROM THE   */     00132500
  1326        /*  PROJECT TABLE.                               */     00132600
  1327        /*************************************************/     00132700
  1328               EXEC SQL SELECT ACTNO, PREQPROJ, PREQACT         00132800
  1329                    INTO PROJ_DATA                              00132900
  1330                    FROM TPREREQ                                00133000
  1331                         WHERE PROJNO = :PROJ_NO;               00133100
  1332                                                                00133200
  1333        /*************************************************/     00133300
  1334        /*  PROJECT IS FINISHED. DELETE IT.              */     00133400
  1335        /*************************************************/     00133500
  1336                                                                00133600
  1337                EXEC SQL DELETE FROM PROJ                       00133700
  1338                     WHERE PROJNO = :PROJ_NO;                   00133800
  ⋮
  1523        END;                                                    00152300
```

Figure 72. Db2 precompiler SYSPRINT output: Source statements section

Notes:

- The left column of sequence numbers, which the Db2 precompiler generates, is for use with the symbol cross-reference listing, the precompiler error messages, and the BIND error messages.
- The right column shows sequence numbers that come from the sequence numbers that are supplied with your source statements.

The following figure shows an example list of symbolic names as it is displayed in the SYSPRINT output.

```
DB2 SQL PRECOMPILER      SYMBOL CROSS-REFERENCE LISTING            PAGE 29

DATA NAMES                     DEFN     REFERENCE

"ACTNO"                        ****     FIELD
                                        1328
"PREQACT"                      ****     FIELD
                                        1328
"PREQPROJ"                     ****     FIELD
                                        1328
"PROJNO"                       ****     FIELD
                                        1331 1338

 ...

PROJ_DATA                      495      CHARACTER(35)
                                        1329
PROJ_NO                        496      CHARACTER(3)
                                        1331 1338
"TPREREQ"                      ****     TABLE
                                        1330 1337
```

Figure 73. Db2 precompiler SYSPRINT output: Symbol cross-reference section

Notes:

DATA NAMES
Identifies the symbolic names that are used in source statements. Names enclosed in double quotation marks (") or apostrophes (') are names of SQL entities such as tables, columns, and authorization IDs. Other names are host variables.

DEFN
Is the number of the line that the precompiler generates to define the name. **** means that the object was not defined, or the precompiler did not recognize the declarations.

REFERENCE
Contains two kinds of information: the symbolic name, which the source program defines, and which lines refer to the symbolic name. If the symbolic name refers to a valid host variable, the list also identifies the data type or the word STRUCTURE.

The following code shows an example summary report of errors as it is displayed in the SYSPRINT output.

```
DB2 SQL PRECOMPILER           STATISTICS

  SOURCE STATISTICS
    SOURCE LINES READ: 1523[1]
    NUMBER OF SYMBOLS: 128[2]
    SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 6432[3]

  THERE WERE 1 MESSAGES FOR THIS PROGRAM.[4]
  THERE WERE 0 MESSAGES SUPPRESSED.[5]
  65536 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.[6]
  RETURN CODE IS 8.[7]
  DSNH104I E LINE 590 COL 64 ILLEGAL SYMBOL: 'X'; VALID SYMBOLS ARE:,FROM[8]
```

Notes:

1. Summary statement that indicates the number of source lines.
2. Summary statement that indicates the number of symbolic names in the symbol table (SQL names and host names).
3. Storage requirement statement that indicates the number of bytes for the symbol table.
4. Summary statement that indicates the number of messages that are printed.
5. Summary statement that indicates the number of errors that are detected but not printed. You might get this statement if you specify the option FLAG.
6. Storage requirement statement that indicates the number of bytes of working storage that are actually used by the Db2 precompiler to process your source statements.
7. Return code 0 = success, 4 = warning, 8 = error, 12 = severe error, and 16 = unrecoverable error.
8. Error messages (this example detects only one error).

Techniques for debugging programs in TSO

Documenting the errors that are identified during testing of a TSO application helps you investigate and correct problems in the program.

The following information can be useful:

- The application plan name of the program
- The input data that is being processed
- The failing SQL statement and its function
- The contents of the SQLCA (SQL communication area) and, if your program accepts dynamic SQL statements, the SQLDA (SQL descriptor area)
- The date and time of day
- The abend code and any error messages

When your program encounters an error that does not result in an abend, it can pass all the required error information to a standard error routine. Online programs might also send an error message to the terminal.

The TSO TEST command

The TSO TEST command is especially useful for debugging assembler programs.

The following example is a command procedure (CLIST) that runs a Db2 application named MYPROG under TSO TEST, and sets an address stop at the entry to the program. The Db2 subsystem name in this example is DB4.

```
PROC 0
TEST 'prefix.SDSNLOAD(DSN)' CP
DSN SYSTEM(DB4)
AT MYPROG.MYPROG.+0 DEFER
GO
RUN  PROGRAM(MYPROG)  LIBRARY('L186331.RUNLIB.LOAD(MYPROG)')
```

Related reference

TEST command (TSO/E Command Reference)

Techniques for debugging programs in IMS

Documenting the errors that are identified during testing of an IMS application helps you investigate and correct problems in the program.

The following information can be useful:

- The application plan name for the program
- The input message that is being processed
- The name of the originating logical terminal
- The failing statement and its function
- The contents of the SQLCA (SQL communication area) and, if your program accepts dynamic SQL statements, the SQLDA (SQL descriptor area)
- The date and time of day
- The PSB name for the program
- The transaction code that the program was processing
- The call function (that is, the name of a DL/I function)
- The contents of the PCB that the program call refers to
- If a DL/I database call was running, the SSAs, if any, that the call used
- The abend completion code, abend reason code, and any dump error messages

When your program encounters an error, it can pass all the required error information to a standard error routine. Online programs can also send an error message to the originating logical terminal.

An interactive program also can send a message to the master terminal operator (MTO) operator giving information about the termination of the program. To do that, the program places the logical terminal name of the master terminal in an express PCB and issues one or more ISRT calls.

Some organizations run a BMP at the end of the day to list all the errors that occurred during the day. If your organization does this, you can send a message by using an express PCB that has its destination set for that BMP.

Batch Terminal Simulator: The Batch Terminal Simulator (BTS) enables you to test IMS application programs. BTS traces application program DL/I calls and SQL statements, and it simulates data communication functions. It can make a TSO terminal appear as an IMS terminal to the terminal operator, which enables the user to interact with the application as though it were an online application. The user can use any application program that is under the user's control to access any database (whether DL/I or Db2) that is under the user's control. Access to Db2 databases requires BTS to operate in batch BMP or TSO BMP mode.

Techniques for debugging programs in CICS

Documenting the errors that are identified during testing of a CICS application helps you investigate and correct problems in the program.

The following information can be useful:

- The application plan name of the program
- The input data that is being processed
- The ID of the originating logical terminal
- The failing SQL statement and its function
- The contents of the SQLCA (SQL communication area) and, if your program accepts dynamic SQL statements, the SQLDA (SQL descriptor area)
- The date and time of day
- Data that is peculiar to CICS that you should record
- Abend code and dump error messages
- Transaction dump, if produced

Using CICS facilities, you can have a printed error record; you can also print the SQLCA and SQLDA contents.

Debugging aids for CICS

CICS provides the following aids to the testing, monitoring, and debugging of application programs:

- **Execution (Command Level) Diagnostic Facility (EDF).** EDF shows CICS commands for all releases of CICS.
- **Abend recovery.** You can use the HANDLE ABEND command to deal with abend conditions. You can use the ABEND command to cause a task to abend.
- **Trace facility.** A trace table can contain entries showing the execution of various CICS commands, SQL statements, and entries that are generated by application programs; you can have these entries written to main storage and, optionally, to an auxiliary storage device.
- **Dump facility.** You can specify areas of main storage to dump onto a sequential data set, either tape or disk, for subsequent offline formatting and printing with a CICS utility program.
- **Journals.** For statistical or monitoring purposes, facilities can create entries in special data sets called journals. The system log is a journal.
- **Recovery.** When an abend occurs, CICS restores certain resources to their original state so that the operator can easily resubmit a transaction for restart. You can use the SYNCPOINT command to subdivide a program so that you only need to resubmit the uncompleted part of a transaction.

CICS execution diagnostic facility

The CICS execution diagnostic facility (EDF) traces SQL statements in an interactive debugging mode, enabling application programmers to test and debug programs online without changing the program or the program preparation procedure.

EDF intercepts the running application program at various points and displays helpful information about the statement type, input and output variables, and any error conditions after the statement executes. It also displays any screens that the application program sends, so that you can converse with the application program during testing just as a user would on a production system.

EDF displays essential information before and after an SQL statement runs, while the task is in EDF mode. This can be a significant aid in debugging CICS transaction programs that contains SQL statements. The SQL information that EDF displays is helpful for debugging programs and for error analysis after an SQL error or warning. Using this facility reduces the amount of work that you need to do to write special error handlers.

EDF before execution

The following figure shows an example of an EDF screen before it executes an SQL statement. The names of the key information fields on this panel are in **boldface**.

```
 TRANSACTION: XC05   PROGRAM: TESTC05    TASK NUMBER: 0000668   DISPLAY:  00
 STATUS:  ABOUT TO EXECUTE COMMAND
 CALL TO RESOURCE MANAGER DSNCSQL
 EXEC SQL INSERT
  DBRM=TESTC05,  STMT=00368, SECT=00004
  IVAR 001: TYPE=CHAR,                 LEN=00007,    IND=000     AT X'03C92810'
            DATA=X'F0F0F9F4F3F4F2'
  IVAR 002: TYPE=CHAR,                 LEN=00007,    IND=000     AT X'03C92817'
            DATA=X'F0F1F3F3F7F5F1'
  IVAR 003: TYPE=CHAR,                 LEN=00004,    IND=000     AT X'03C9281E'
            DATA=X'E7C3F0F5'
  IVAR 004: TYPE=CHAR,                 LEN=00040,    IND=000     AT X'03C92822'
            DATA=X'E3C5E2E3C3F0F540E2C9D4D7D3C540C4C2F240C9D5E2C5D9E3404040'...
  IVAR 005: TYPE=SMALLINT,             LEN=00002,    IND=000     AT X'03C9284A'
            DATA=X'0001'

 OFFSET:X'001ECE'    LINE:UNKNOWN        EIBFN=X'1002'


ENTER:  CONTINUE
PF1 : UNDEFINED          PF2 : UNDEFINED          PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD     PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY   PF11: UNDEFINED          PF12: ABEND USER TASK
```

Figure 74. EDF screen before a Db2 SQL statement

The Db2 SQL information in this screen is as follows:

- EXEC SQL *statement type*

  This is the type of SQL statement to execute. The SQL statement can be any valid SQL statement.

- DBRM=*dbrm name*

  The name of the database request module (DBRM) that is currently processing. The DBRM, created by the Db2 precompiler, contains information about an SQL statement.

- STMT=*statement number*

  This is the Db2 precompiler-generated statement number. The source and error message listings from the precompiler use this statement number, and you can use the statement number to determine which statement is processing. This number is a source line counter that includes host language statements. A statement number that is greater than 32 767 displays as 0.

- SECT=*section number*

  The section number of the plan that the SQL statement uses.

SQL statements that contain input host variables

The IVAR (input host variables) section and its attendant fields appear only when the executing statement contains input host variables.

The host variables section includes the variables from predicates, the values used for inserting or updating, and the text of dynamic SQL statements that are being prepared. The address of the input variable is AT X'nnnnnnnn'.

Additional host variable information:

- TYPE=*data type*

  Specifies the data type for this host variable. The basic data types include character string, graphic string, binary integer, floating-point, decimal, date, time, and timestamp.

- LEN=*length*

Specifies the length of the host variable.

- IND=*indicator variable status number*

  Specifies the indicator variable that is associated with this particular host variable. A value of zero indicates that no indicator variable exists. If the value for the selected column is null, Db2 puts a negative value in the indicator variable for this host variable.

- DATA=*host variable data*

  Specifies the data, displayed in hexadecimal format, that is associated with this host variable. If the data exceeds what can display on a single line, three periods (...) appear at the far right to indicate that more data is present.

EDF after execution

The following figure shows an example of the first EDF screen that is displayed after the executing an SQL statement. The names of the key information fields on this panel are in **boldface**.

```
  TRANSACTION: XC05  PROGRAM: TESTC05    TASK NUMBER: 0000698   DISPLAY:  00
  STATUS: COMMAND EXECUTION COMPLETE
  CALL TO RESOURCE MANAGER DSNCSQL
  EXEC SQL FETCH                            P.AUTH=SYSADM  , S.AUTH=
  PLAN=TESTC05, DBRM=TESTC05,  STMT=00346, SECT=00001
  SQL COMMUNICATION AREA:
   SQLCABC     = 136                                           AT X'03C92789'
   SQLCODE     = 000                                           AT X'03C9278D'
   SQLERRML    = 000                                           AT X'03C92791'
   SQLERRMC    = ''                                            AT X'03C92793'
   SQLERRP     = 'DSN'                                         AT X'03C927D9'
   SQLERRD(1-6) = 000,  000, 00000, -1, 00000,  000            AT X'03C927E1'
   SQLWARN(0-A) = '_ _ _ _ _ _ _ _ _ _ _'                      AT X'03C927F9'
   SQLSTATE    = 00000                                         AT X'03C92804'
+ OVAR 001: TYPE=INTEGER,            LEN=00004,    IND=000     AT X'03C920A0'
            DATA=X'00000001'
  OFFSET:X'001D14'   LINE:UNKNOWN        EIBFN=X'1802'


ENTER:   CONTINUE
PF1  : UNDEFINED          PF2 : UNDEFINED           PF3 : END EDF SESSION
PF4  : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7  : SCROLL BACK        PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10:  PREVIOUS DISPLAY   PF11: UNDEFINED           PF12: ABEND USER TASK
```

Figure 75. EDF screen after a Db2 SQL statement

The Db2 SQL information in this screen is as follows:

- P.AUTH=*primary authorization ID*

  The primary Db2 authorization ID.

- S.AUTH=*secondary authorization ID*

  The secondary authorization ID. If the RACF list of group options is not active, Db2 uses the connected group name that the CICS attachment facility supplies as the secondary authorization ID. If the RACF list of group options is active, Db2 ignores the connected group name that the CICS attachment facility supplies, but the value is displayed in the Db2 list of secondary authorization IDs.

- PLAN=*plan name*

  The name of the plan that is currently running. The PLAN represents the control structure that is produced during the bind process and that is used by Db2 to process SQL statements that are encountered while the application is running.

- SQL Communication Area (SQLCA)

  Information in the SQLCA. The SQLCA contains information about errors, if any occur. Db2 uses the SQLCA to give an application program information about the executing SQL statements.

Plus signs (+) on the left of the screen indicate that you can see additional EDF output by using PF keys to scroll the screen forward or back.

The OVAR (output host variables) section and its attendant fields are displayed only when the executing statement returns output host variables.

The following figure contains the rest of the EDF output for this example.

```
 TRANSACTION: XC05   PROGRAM: TESTC05    TASK NUMBER: 0000698   DISPLAY:  00
 STATUS:  COMMAND EXECUTION COMPLETE
 CALL TO RESOURCE MANAGER DSNCSQL
+ OVAR 002: TYPE=CHAR,                 LEN=00008,    IND=000     AT X'03C920B0'
            DATA=X'C8F3E3E3C1C2D3C5'
  OVAR 003: TYPE=CHAR,                 LEN=00040,    IND=000     AT X'03C920B8'
            DATA=X'C9D5C9E3C9C1D340D3D6C1C44040404040404040404040404040404040'...









 OFFSET:X'001D14'    LINE:UNKNOWN        EIBFN=X'1802'


ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : UNDEFINED           PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: UNDEFINED           PF12: ABEND USER TASK
```

Figure 76. EDF screen after a Db2 SQL statement, continued

The attachment facility automatically displays SQL information while in the EDF mode. (You can start EDF as outlined in the appropriate CICS application programmer's reference manual.) If this information is not displayed, contact the person that is responsible for installing and migrating Db2.

Related concepts

Data types of columns
When you create a Db2 table, you define each column to have a specific data type. The data type of a column determines what you can and cannot do with the column.

Indicator variables, arrays, and structures
An indicator variable is associated with a particular host variable. Each indicator variable contains a small integer value that indicates some information about the associated host variable. Indicator arrays and structures serve the same purpose for host-variable arrays and structures.

Related information

CICS debugging aids (CICS Transaction Server for z/OS)

Finding a violated referential or check constraint

When you receive an SQL error because of a constraint violation, look at the SQLCA for specific information.

About this task

Question: When a referential or check constraint has been violated, how do I determine which one it is?

Answer: When you receive an SQL error because of a constraint violation, print out the SQLCA. You can use the DSNTIAR routine to format the SQLCA for you. Check the SQL error message insertion text (SQLERRM) for the name of the constraint. For information about possible violations, see SQLCODEs -530 through -548.

Related concepts

SQL error codes (Db2 Codes)

Related tasks

Displaying SQLCA fields by calling DSNTIAR

If you use the SQLCA to check whether an SQL statement executed successfully, your program needs to read the data in the appropriate SQLCA fields. One easy way to read these fields is to use the assembler subroutine DSNTIAR.

Chapter 12. Sample data and applications supplied with Db2 for z/OS

You can use sample applications that are included with Db2 for z/OS to learn about how to program applications that take advantage Db2 capabilities. Db2 also provides models for your own situations.

To prepare and run the supplied sample applications, use the JCL in *prefix*.SDSNSAMP as a model:

Related reference

Db2 sample tables (Introduction to Db2 for z/OS)

Db2 sample tables

Much of the Db2 information refers to or relies on the Db2 sample tables. As a group, the tables include information that describes employees, departments, projects, and activities, and they make up a sample application that exemplifies many of the features of Db2.

GUPI

The sample storage group, databases, table spaces, tables, and views are created when you run the installation sample jobs DSNTEJ1 and DSNTEJ7. Db2 sample objects that include LOBs are created in job DSNTEJ7. All other sample objects are created in job DSNTEJ1. The CREATE INDEX statements for the sample tables are not shown here; they, too, are created by the DSNTEJ1 and DSNTEJ7 sample jobs.

Authorization on all sample objects is given to PUBLIC in order to make the sample programs easier to run. You can review the contents of any table by executing an SQL statement, for example SELECT * FROM DSN8B10.PROJ. For convenience in interpreting the examples, the department and employee tables are listed in full.

GUPI

Related concepts

Phase 1: Creating and loading sample tables (Db2 Installation and Migration)

Activity table (DSN8B10.ACT)

The activity table describes the activities that can be performed during a project.

GUPI

The activity table resides in database DSN8D11A and is created with the following statement:

```
CREATE TABLE DSN8B10.ACT
      (ACTNO    SMALLINT       NOT NULL,
       ACTKWD   CHAR(6)        NOT NULL,
       ACTDESC  VARCHAR(20)    NOT NULL,
       PRIMARY KEY (ACTNO)             )
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the activity table

The following table shows the content of the columns in the activity table.

Table 156. Columns of the activity table

Column	Column name	Description
1	ACTNO	Activity ID (the primary key)
2	ACTKWD	Activity keyword (up to six characters)
3	ACTDESC	Activity description

The activity table has the following indexes.

Table 157. Indexes of the activity table

Name	On column	Type of index
DSN8B10.XACT1	ACTNO	Primary, ascending
DSN8B10.XACT2	ACTKWD	Unique, ascending

Relationship to other tables

The activity table is a parent table of the project activity table, through a foreign key on column ACTNO.

Department table (DSN8B10.DEPT)

The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

GUPI

The department table resides in table space DSN8D11A.DSN8S11D and is created with the following statement:

```
CREATE TABLE DSN8B10.DEPT
      (DEPTNO    CHAR(3)            NOT NULL,
       DEPTNAME  VARCHAR(36)        NOT NULL,
       MGRNO     CHAR(6)                    ,
       ADMRDEPT  CHAR(3)            NOT NULL,
       LOCATION  CHAR(16)                   ,
       PRIMARY KEY (DEPTNO)                 )
  IN DSN8D11A.DSN8S11D
  CCSID EBCDIC;
```

Because the department table is self-referencing, and also is part of a cycle of dependencies, its foreign keys must be added later with the following statements:

```
ALTER TABLE DSN8B10.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8B10.DEPT
              ON DELETE CASCADE;

ALTER TABLE DSN8B10.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8B10.EMP
              ON DELETE SET NULL;
```

GUPI

Content of the department table

The following table shows the content of the columns in the department table.

Table 158. Columns of the department table

Column	Column name	Description
1	DEPTNO	Department ID, the primary key.

Column	Column name	Description
2	DEPTNAME	A name that describes the general activities of the department.
3	MGRNO	Employee number (EMPNO) of the department manager.
4	ADMRDEPT	ID of the department to which this department reports; the department at the highest level reports to itself.
5	LOCATION	The remote location name.

Table 158. Columns of the department table (continued)

The following table shows the indexes of the department table.

Table 159. Indexes of the department table

Name	On column	Type of index
DSN8B10.XDEPT1	DEPTNO	Primary, ascending
DSN8B10.XDEPT2	MGRNO	Ascending
DSN8B10.XDEPT3	ADMRDEPT	Ascending

The following table shows the content of the department table.

Table 160. DSN8B10.DEPT: department table

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	----------------
B01	PLANNING	000020	A00	----------------
C01	INFORMATION CENTER	000030	A00	----------------
D01	DEVELOPMENT CENTER	------	A00	----------------
E01	SUPPORT SERVICES	000050	A00	----------------
D11	MANUFACTURING SYSTEMS	000060	D01	----------------
D21	ADMINISTRATION SYSTEMS	000070	D01	----------------
E11	OPERATIONS	000090	E01	----------------
E21	SOFTWARE SUPPORT	000100	E01	----------------
F22	BRANCH OFFICE F2	------	E01	----------------
G22	BRANCH OFFICE G2	------	E01	----------------
H22	BRANCH OFFICE H2	------	E01	----------------
I22	BRANCH OFFICE I2	------	E01	----------------
J22	BRANCH OFFICE J2	------	E01	----------------

The LOCATION column contains null values until sample job DSNTEJ6 updates this column with the location name.

Relationship to other tables

The department table is self-referencing: the value of the administering department must be a valid department ID.

The department table is a parent table of the following :

- The employee table, through a foreign key on column WORKDEPT
- The project table, through a foreign key on column DEPTNO

The department table is a dependent of the employee table, through its foreign key on column MGRNO.

Employee table (DSN8B10.EMP)

The sample employee table identifies all employees by an employee number and lists basic personnel information.

GUPI The employee table resides in the partitioned table space DSN8D11A.DSN8S11E. Because this table has a foreign key that references DEPT, that table and the index on its primary key must be created first. Then EMP is created with the following statement:

```
CREATE TABLE DSN8B10.EMP
      (EMPNO     CHAR(6)                         NOT NULL,
       FIRSTNME  VARCHAR(12)                     NOT NULL,
       MIDINIT   CHAR(1)                         NOT NULL,
       LASTNAME  VARCHAR(15)                     NOT NULL,
       WORKDEPT  CHAR(3)                                 ,
       PHONENO   CHAR(4)         CONSTRAINT NUMBER CHECK
                 (PHONENO >= '0000' AND
                  PHONENO <= '9999')                     ,
       HIREDATE  DATE                                    ,
       JOB       CHAR(8)                                 ,
       EDLEVEL   SMALLINT                                ,
       SEX       CHAR(1)                                 ,
       BIRTHDATE DATE                                    ,
       SALARY    DECIMAL(9,2)                            ,
       BONUS     DECIMAL(9,2)                            ,
       COMM      DECIMAL(9,2)                            ,
       PRIMARY KEY (EMPNO)                               ,
       FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8B10.DEPT
                 ON DELETE SET NULL                      )
  EDITPROC  DSN8EAE1
  IN DSN8D11A.DSN8S11E
  CCSID EBCDIC;
```

GUPI

Content of the employee table

The following table shows the type of content of each of the columns in the employee table. The table has a check constraint, NUMBER, which checks that the four-digit phone number is in the numeric range 0000 to 9999.

Table 161. Columns of the employee table

Column	Column name	Description
1	EMPNO	Employee number (the primary key)
2	FIRSTNME	First name of employee
3	MIDINIT	Middle initial of employee
4	LASTNAME	Last name of employee
5	WORKDEPT	ID of department in which the employee works
6	PHONENO	Employee telephone number
7	HIREDATE	Date of hire
8	JOB	Job held by the employee
9	EDLEVEL	Number of years of formal education
10	SEX	Sex of the employee (M or F)

Table 161. Columns of the employee table (continued)		
Column	**Column name**	**Description**
11	BIRTHDATE	Date of birth
12	SALARY	Yearly salary in dollars
13	BONUS	Yearly bonus in dollars
14	COMM	Yearly commission in dollars

The following table shows the indexes of the employee table.

Table 162. Indexes of the employee table

Name	On column	Type of index
DSN8B10.XEMP1	EMPNO	Primary, partitioned, ascending
DSN8B10.XEMP2	WORKDEPT	Ascending

The following table shows the first half (left side) of the content of the employee table. (Table 164 on page 980 shows the remaining content (right side) of the employee table.)

Table 163. Left half of DSN8B10.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of " " rather than null.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10
000030	SALLY	A	KWAN	C01	4738	1975-04-05
000050	JOHN	B	GEYER	E01	6789	1949-08-17
000060	IRVING	F	STERN	D11	6423	1973-09-14
000070	EVA	D	PULASKI	D21	7831	1980-09-30
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16
000120	SEAN		O'CONNELL	A00	2167	1963-12-05
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15
000150	BRUCE		ADAMSON	D11	4510	1972-02-12
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07
000190	JAMES	H	WALKER	D11	2986	1974-07-26
000200	DAVID		BROWN	D11	4501	1966-03-03
000210	WILLIAM	T	JONES	D11	0942	1979-04-11
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21

Table 163. Left half of DSN8B10.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of " " rather than null. *(continued)*

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1969-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5698	1947-05-05
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01
200120	GREG		ORLANDO	A00	2167	1972-05-05
200140	KIM	N	NATZ	C01	1793	1976-12-15
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15
200220	REBA	K	JOHN	D11	0672	1968-08-29
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12
200330	HELENA		WONG	E21	2103	1976-02-23
200340	ROY	R	ALONZO	E21	5698	1947-05-05

(Table 163 on page 979 shows the first half (right side) of the content of employee table.)

Table 164. Right half of DSN8B10.EMP: employee table

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000010)	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
(000020)	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
(000030)	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
(000050)	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
(000060)	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
(000070)	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
(000090)	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
(000100)	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
(000110)	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
(000120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(000130)	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(000150)	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
(000160)	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
(000170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(000180)	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
(000190)	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
(000200)	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
(000210)	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
(000220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(000230)	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
(000240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(000250)	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
(000260)	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
(000270)	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
(000280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(000290)	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
(000300)	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
(000310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(000320)	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
(000330)	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
(000340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
(200010)	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
(200120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(200140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(200170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(200220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(200240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(200280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(200310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(200330)	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
(200340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

Table 164. Right half of DSN8B10.EMP: employee table (continued)

Relationship to other tables

The employee table is a parent table of:

- The department table, through a foreign key on column MGRNO
- The project table, through a foreign key on column RESPEMP

The employee table is a dependent of the department table, through its foreign key on column WORKDEPT.

Employee photo and resume table (DSN8B10.EMP_PHOTO_RESUME)

The sample employee photo and resume table complements the employee table.

GUPI Each row of the photo and resume table contains a photo of the employee, in two formats, and the employee's resume. The photo and resume table resides in table space .DSN8S11B. The following statement creates the table:

```
CREATE TABLE  DSN8B10.EMP_PHOTO_RESUME
               (EMPNO      CHAR(06) NOT NULL,
                EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
                PSEG_PHOTO BLOB(500K),
                BMP_PHOTO  BLOB(100K),
                RESUME        CLOB(5K))
                PRIMARY KEY  (EMPNO)
        IN  DSN8D11L.DSN8S11B
        CCSID  EBCDIC;
```

Db2 requires an auxiliary table for each LOB column in a table. The following statements define the auxiliary tables for the three LOB columns in DSN8B10.EMP_PHOTO_RESUME:

```
CREATE AUX TABLE DSN8B10.AUX_BMP_PHOTO
               IN DSN8D11L.DSN8S11M
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8B10.AUX_PSEG_PHOTO
               IN DSN8D11L.DSN8S11L
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN PSEG_PHOTO;

CREATE AUX TABLE DSN8B10.AUX_EMP_RESUME
               IN DSN8D11L.DSN8S11N
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN RESUME;
```

GUPI

Content of the employee photo and resume table

The following table shows the content of the columns in the employee photo and resume table.

Table 165. Columns of the employee photo and resume table

Column	Column name	Description
1	EMPNO	Employee ID (the primary key).
2	EMP_ROWID	Row ID to uniquely identify each row of the table. Db2 supplies the values of this column.
3	PSEG_PHOTO	Employee photo, in PSEG format.
4	BMP_PHOTO	Employee photo, in BMP format.
5	RESUME	Employee resume.

The following table shows the indexes for the employee photo and resume table.

Table 166. Indexes of the employee photo and resume table

Name	On column	Type of index
DSN8B10.XEMP_PHOTO_RESUME	EMPNO	Primary, ascending

The following table shows the indexes for the auxiliary tables that support the employee photo and resume table.

Table 167. Indexes of the auxiliary tables for the employee photo and resume table

Name	On table	Type of index
DSN8B10.XAUX_BMP_PHOTO	DSN8B10.AUX_BMP_PHOTO	Unique
DSN8B10.XAUX_PSEG_PHOTO	DSN8B10.AUX_PSEG_PHOTO	Unique
DSN8B10.XAUX_EMP_RESUME	DSN8B10.AUX_EMP_RESUME	Unique

Relationship to other tables

The employee photo and resume table is a parent table of the project table, through a foreign key on column RESPEMP.

Project table (DSN8B10.PROJ)

The sample project table describes each project that the business is currently undertaking. Data that is contained in each row of the table includes the project number, name, person responsible, and schedule dates.

The project table resides in database DSN8D11A. Because this table has foreign keys that reference DEPT and EMP, those tables and the indexes on their primary keys must be created first. Then PROJ is created with the following statement:

GUPI

```
CREATE TABLE DSN8B10.PROJ
               (PROJNO   CHAR(6) PRIMARY KEY NOT NULL,
                PROJNAME VARCHAR(24)    NOT NULL WITH DEFAULT
                  'PROJECT NAME UNDEFINED',
                DEPTNO   CHAR(3)        NOT NULL REFERENCES
                  DSN8B10.DEPT ON DELETE RESTRICT,
                RESPEMP  CHAR(6)        NOT NULL REFERENCES
                  DSN8B10.EMP ON DELETE RESTRICT,
                PRSTAFF  DECIMAL(5, 2)          ,
                PRSTDATE DATE                   ,
                PRENDATE DATE                   ,
                MAJPROJ  CHAR(6))
        IN DSN8D11A.DSN8S11P
        CCSID EBCDIC;
```

Because the project table is self-referencing, the foreign key for that constraint must be added later with the following statement:

```
ALTER TABLE DSN8B10.PROJ
      FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8B10.PROJ
              ON DELETE CASCADE;
```

GUPI

Content of the project table

The following table shows the content of the columns of the project table.

Table 168. Columns of the project table

Column	Column name	Description
1	PROJNO	Project ID (the primary key)
2	PROJNAME	Project name

Column	Column name	Description
3	DEPTNO	ID of department responsible for the project
4	RESPEMP	ID of employee responsible for the project
5	PRSTAFF	Estimated mean number of persons that are needed between PRSTDATE and PRENDATE to complete the whole project, including any subprojects
6	PRSTDATE	Estimated project start date
7	PRENDATE	Estimated project end date
8	MAJPROJ	ID of any project of which this project is a part

Table 168. Columns of the project table (continued)

The following table shows the indexes for the project table:

Table 169. Indexes of the project table

Name	On column	Type of index
DSN8B10.XPROJ1	PROJNO	Primary, ascending
DSN8B10.XPROJ2	RESPEMP	Ascending

Relationship to other tables

The table is self-referencing: a non-null value of MAJPROJ must be a valid project number. The table is a parent table of the project activity table, through a foreign key on column PROJNO. It is a dependent of the following tables:

- The department table, through its foreign key on DEPTNO
- The employee table, through its foreign key on RESPEMP

Project activity table (DSN8B10.PROJACT)

The sample project activity table lists the activities that are performed for each project.

The project activity table resides in database DSN8D11A. Because this table has foreign keys that reference PROJ and ACT, those tables and the indexes on their primary keys must be created first. Then PROJACT is created with the following statement:

GUPI

```
CREATE TABLE DSN8B10.PROJACT
      (PROJNO    CHAR(6)                         NOT NULL,
       ACTNO     SMALLINT                        NOT NULL,
       ACSTAFF   DECIMAL(5,2)                            ,
       ACSTDATE  DATE                            NOT NULL,
       ACENDATE  DATE                                    ,
       PRIMARY KEY (PROJNO, ACTNO, ACSTDATE),
       FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8B10.PROJ
                                 ON DELETE RESTRICT,
       FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8B10.ACT
                                 ON DELETE RESTRICT)
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the project activity table

The following table shows the content of the columns of the project activity table.

Table 170. Columns of the project activity table

Column	Column name	Description
1	PROJNO	Project ID
2	ACTNO	Activity ID
3	ACSTAFF	Estimated mean number of employees that are needed to staff the activity
4	ACSTDATE	Estimated activity start date
5	ACENDATE	Estimated activity completion date

The following table shows the index of the project activity table:

Table 171. Index of the project activity table

Name	On columns	Type of index
DSN8B10.XPROJAC1	PROJNO, ACTNO, ACSTDATE	primary, ascending

Relationship to other tables

The project activity table is a parent table of the employee to project activity table, through a foreign key on columns PROJNO, ACTNO, and EMSTDATE. It is a dependent of the following tables:

- The activity table, through its foreign key on column ACTNO
- The project table, through its foreign key on column PROJNO

Related reference

Activity table (DSN8B10.ACT) (Introduction to Db2 for z/OS)
Project table (DSN8B10.PROJ) (Introduction to Db2 for z/OS)

Employee-to-project activity table (DSN8B10.EMPPROJACT)

The sample employee-to-project activity table identifies the employee who performs an activity for a project, tells the proportion of the employee's time that is required, and gives a schedule for the activity.

GUPI

The employee-to-project activity table resides in database DSN8D11A. Because this table has foreign keys that reference EMP and PROJACT, those tables and the indexes on their primary keys must be created first. Then EMPPROJACT is created with the following statement:

```
CREATE TABLE DSN8B10.EMPPROJACT
      (EMPNO     CHAR(6)                          NOT NULL,
       PROJNO    CHAR(6)                          NOT NULL,
       ACTNO     SMALLINT                         NOT NULL,
       EMPTIME   DECIMAL(5,2)                             ,
       EMSTDATE  DATE                                     ,
       EMENDATE  DATE                                     ,
       FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
                 REFERENCES DSN8B10.PROJACT
                                ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8B10.EMP
                                ON DELETE RESTRICT)
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the employee-to-project activity table

The following table shows the content of the columns in the employee-to-project activity table.

Table 172. Columns of the employee-to-project activity table

Column	Column name	Description
1	EMPNO	Employee ID number
2	PROJNO	Project ID of the project
3	ACTNO	ID of the activity within the project
4	EMPTIME	A proportion of the employee's full time (between 0.00 and 1.00) that is to be spent on the activity
5	EMSTDATE	Date the activity starts
6	EMENDATE	Date the activity ends

The following table shows the indexes for the employee-to-project activity table:

Table 173. Indexes of the employee-to-project activity table

Name	On columns	Type of index
DSN8B10.XEMPPROJACT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Unique, ascending
DSN8B10.XEMPPROJACT2	EMPNO	Ascending

Relationship to other tables

The employee-to-project activity table is a dependent of the following tables:

- The employee table, through its foreign key on column EMPNO
- The project activity table, through its foreign key on columns PROJNO, ACTNO, and EMSTDATE.

Related reference

Employee table (DSN8B10.EMP) (Introduction to Db2 for z/OS)
Project activity table (DSN8B10.PROJACT) (Introduction to Db2 for z/OS)

Unicode sample table (DSN8B10.DEMO_UNICODE)

The Unicode sample table is used to verify that data conversions to and from EBCDIC and Unicode are working as expected.

GUPI

The table resides in database DSN8D11A, and is defined with the following statement:

```
CREATE TABLE DSN8B10.DEMO_UNICODE
           (LOWER_A_TO_Z      CHAR(26)                      ,
            UPPER_A_TO_Z      CHAR(26)                      ,
            ZERO_TO_NINE      CHAR(10)                      ,
            X00_TO_XFF    VARCHAR(256)        FOR BIT DATA)
   IN DSN8D81E.DSN8S81U
   CCSID UNICODE;
```

GUPI

Content of the Unicode sample table

The following table shows the content of the columns in the Unicode sample table:

Table 174. Columns of the Unicode sample table		
Column	**Column Name**	**Description**
1	LOWER_A_TO_Z	Array of characters, 'a' to 'z'
2	UPPER_A_TO_Z	Array of characters, 'A' to 'Z'
3	ZERO_TO_NINE	Array of characters, '0' to '9'
4	X00_TO_XFF	Array of characters, x'00' to x'FF'

This table has no indexes.

Relationship to other tables

This table has no relationship to other tables.

Relationships among the sample tables

Relationships among the sample tables are established by foreign keys in dependent tables that reference primary keys in parent tables.

The following figure shows relationships among the sample tables. You can find descriptions of the columns with the descriptions of the tables.

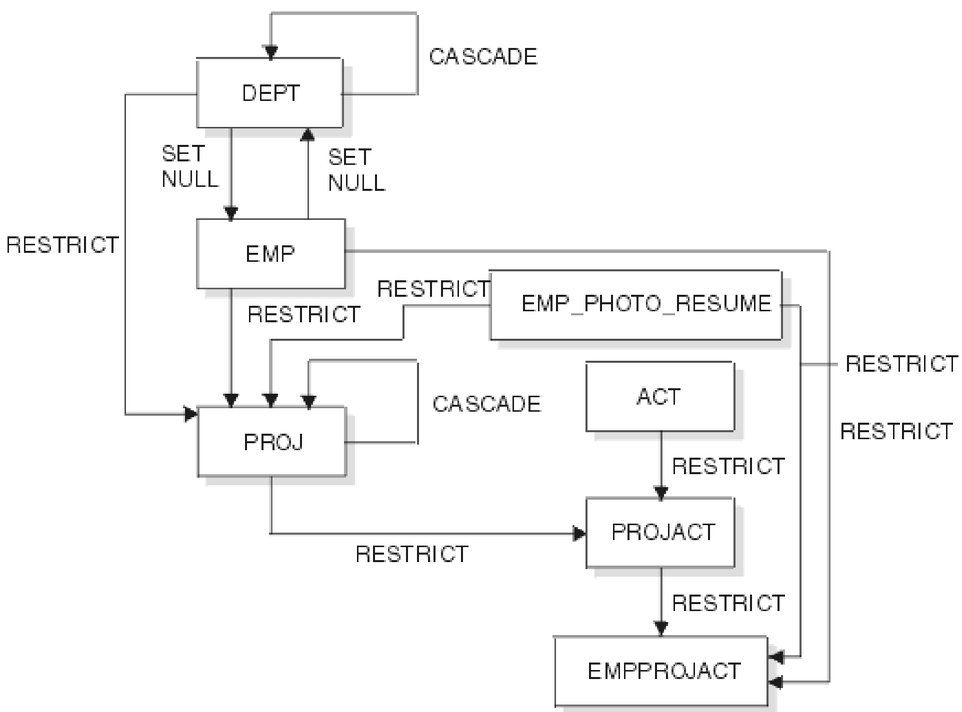


Figure 77. Relationships among tables in the sample application

Related reference

Activity table (DSN8B10.ACT) (Introduction to Db2 for z/OS)
Department table (DSN8B10.DEPT) (Introduction to Db2 for z/OS)
Employee photo and resume table (DSN8B10.EMP_PHOTO_RESUME) (Introduction to Db2 for z/OS)
Employee table (DSN8B10.EMP) (Introduction to Db2 for z/OS)
Employee-to-project activity table (DSN8B10.EMPPROJACT) (Introduction to Db2 for z/OS)
Project activity table (DSN8B10.PROJACT) (Introduction to Db2 for z/OS)
Project table (DSN8B10.PROJ) (Introduction to Db2 for z/OS)

Unicode sample table (DSN8B10.DEMO_UNICODE) (Introduction to Db2 for z/OS)

Views on the sample tables

Db2 creates a number of views on the sample tables for use in the sample applications.

The following table indicates the tables on which each view is defined and the sample applications that use the view. All view names have the qualifier DSN8B10.

Table 175. Views on sample tables

View name	On tables or views	Used in application
VDEPT	DEPT	Organization Project
VHDEPT	DEPT	Distributed organization
VEMP	EMP	Distributed organization Organization Project
VPROJ	PROJ	Project
VACT	ACT	Project
VPROJACT	PROJACT	Project
VEMPPROJACT	EMPPROJACT	Project
VDEPMG1	DEPT EMP	Organization
VEMPDPT1	DEPT EMP	Organization
VASTRDE1	DEPT	
VASTRDE2	VDEPMG1 EMP	Organization
VPROJRE1	PROJ EMP	Project
VPSTRDE1	VPROJRE1 VPROJRE2	Project
VPSTRDE2	VPROJRE1	Project
VFORPLA	VPROJRE1 EMPPROJACT	Project
VSTAFAC1	PROJACT ACT	Project

Table 175. Views on sample tables (continued)		
View name	**On tables or views**	**Used in application**
VSTAFAC2	EMPPROJACT ACT EMP	Project
VPHONE	EMP DEPT	Phone
VEMPLP	EMP	Phone

GUPI

The following SQL statement creates the view named VDEPT.

```
CREATE VIEW DSN8B10.VDEPT
   AS SELECT ALL      DEPTNO   ,
                      DEPTNAME,
                      MGRNO    ,
                      ADMRDEPT
   FROM DSN8B10.DEPT;
```

The following SQL statement creates the view named VHDEPT.

```
 CREATE VIEW DSN8B10.VHDEPT
    AS SELECT ALL      DEPTNO   ,
                       DEPTNAME,
                       MGRNO    ,
                       ADMRDEPT,
                       LOCATION
    FROM DSN8B10.DEPT;
```

The following SQL statement creates the view named VEMP.

```
CREATE VIEW DSN8B10.VEMP
   AS SELECT ALL      EMPNO    ,
                      FIRSTNME,
                      MIDINIT  ,
                      LASTNAME,
                      WORKDEPT
   FROM DSN8B10.EMP;
```

The following SQL statement creates the view named VPROJ.

```
CREATE VIEW DSN8B10.VPROJ
   AS SELECT ALL
             PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
             PRSTDATE, PRENDATE, MAJPROJ
   FROM DSN8B10.PROJ ;
```

The following SQL statement creates the view named VACT.

```
CREATE VIEW DSN8B10.VACT
      AS SELECT ALL   ACTNO    ,
                      ACTKWD   ,
                      ACTDESC
           FROM DSN8B10.ACT ;
```

The following SQL statement creates the view named VPROJACT.

```
CREATE VIEW DSN8B10.VPROJACT
      AS SELECT ALL
            PROJNO,ACTNO, ACSTAFF, ACSTDATE, ACENDATE
            FROM DSN8B10.PROJACT ;
```

The following SQL statement creates the view named VEMPPROJACT.

```
CREATE VIEW DSN8B10.VEMPPROJACT
      AS SELECT ALL
          EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
          FROM DSN8B10.EMPPROJACT ;
```

The following SQL statement creates the view named VDEPMG1.

```
CREATE VIEW DSN8B10.VDEPMG1
      (DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT,
       LASTNAME, ADMRDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT,
        LASTNAME, ADMRDEPT
        FROM DSN8B10.DEPT LEFT OUTER JOIN DSN8B10.EMP
        ON MGRNO = EMPNO ;
```

The following SQL statement creates the view named VEMPDPT1.

```
CREATE VIEW DSN8B10.VEMPDPT1
      (DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
       LASTNAME, WORKDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
       LASTNAME, WORKDEPT
        FROM DSN8B10.DEPT RIGHT OUTER JOIN DSN8B10.EMP
        ON WORKDEPT = DEPTNO ;
```

The following SQL statement creates the view named VASTRDE1.

```
CREATE VIEW DSN8B10.VASTRDE1
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
        D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
        D1.LASTNAME, '1',
        D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
        D2.LASTNAME
        FROM DSN8B10.VDEPMG1 D1, DSN8B10.VDEPMG1 D2
        WHERE D1.DEPTNO = D2.ADMRDEPT ;
```

The following SQL statement creates the view named VASTRDE2.

```
CREATE VIEW DSN8B10.VASTRDE2
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
         D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
         D1.LASTNAME,'2',
         D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
         E2.LASTNAME
         FROM DSN8B10.VDEPMG1 D1, DSN8B10.EMP E2
         WHERE D1.DEPTNO = E2.WORKDEPT;
```

The following figure shows the SQL statement that creates the view named VPROJRE1.

```
CREATE VIEW DSN8B10.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
   LASTNAME,MAJPROJ)
   AS SELECT ALL
      PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,
      LASTNAME,MAJPROJ
     FROM DSN8B10.PROJ, DSN8B10.EMP
     WHERE RESPEMP = EMPNO ;
```

Figure 78. VPROJRE1

The following SQL statement creates the view named VPSTRDE1.

```
CREATE VIEW DSN8B10.VPSTRDE1
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
```

```
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
        P2.LASTNAME
     FROM DSN8B10.VPROJRE1 P1,
       DSN8B10.VPROJRE1 P2
     WHERE P1.PROJNO = P2.MAJPROJ ;
```

The following SQL statement creates the view named VPSTRDE2.

```
CREATE VIEW DSN8B10.VPSTRDE2
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME
     FROM DSN8B10.VPROJRE1 P1
       WHERE NOT EXISTS
         (SELECT * FROM DSN8B10.VPROJRE1 P2
           WHERE P1.PROJNO = P2.MAJPROJ) ;
```

The following SQL statement creates the view named VFORPLA.

```
CREATE VIEW DSN8B10.VFORPLA
  (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
   AS SELECT ALL
      F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
      MIDINIT, LASTNAME
     FROM DSN8B10.VPROJRE1 F1 LEFT OUTER JOIN DSN8B10.EMPPROJACT F2
     ON F1.PROJNO = F2.PROJNO;
```

The following SQL statement creates the view named VSTAFAC1.

```
CREATE VIEW DSN8B10.VSTAFAC1
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE,ENDATE, TYPE)
   AS SELECT ALL
         PA.PROJNO, PA.ACTNO, AC.ACTDESC,' ', ' ', ' ', ' ',
         PA.ACSTAFF, PA.ACSTDATE,
         PA.ACENDATE,'1'
     FROM DSN8B10.PROJACT PA, DSN8B10.ACT AC
     WHERE PA.ACTNO = AC.ACTNO ;
```

The following SQL statement creates the view named VSTAFAC2.

```
CREATE VIEW DSN8B10.VSTAFAC2
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE, ENDATE, TYPE)
   AS SELECT ALL
         EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO,EM.FIRSTNME,
         EM.MIDINIT, EM.LASTNAME, EP.EMPTIME, EP.EMSTDATE,
         EP.EMENDATE,'2'
     FROM DSN8B10.EMPPROJACT EP, DSN8B10.ACT AC, DSN8B10.EMP EM
     WHERE EP.ACTNO = AC.ACTNO  AND EP.EMPNO = EM.EMPNO ;
```

The following SQL statement creates the view named VPHONE.

```
CREATE VIEW DSN8B10.VPHONE
              (LASTNAME,
               FIRSTNAME,
               MIDDLEINITIAL,
               PHONENUMBER,
               EMPLOYEENUMBER,
               DEPTNUMBER,
               DEPTNAME)
   AS SELECT ALL      LASTNAME,
                      FIRSTNME,
                      MIDINIT ,
                      VALUE(PHONENO,'    '),
                      EMPNO,
                      DEPTNO,
                      DEPTNAME
```

```
    FROM DSN8B10.EMP, DSN8B10.DEPT
    WHERE WORKDEPT = DEPTNO;
```

The following SQL statement creates the view named VEMPLP.

```
CREATE VIEW DSN8B10.VEMPLP
              (EMPLOYEENUMBER,
               PHONENUMBER)
   AS SELECT ALL      EMPNO    ,
                      PHONENO
   FROM DSN8B10.EMP ;
```

GUPI

Storage of sample application tables

Normally, related data is stored in the same database.

The following figure shows how the sample tables are related to databases and storage groups. Two databases are used to illustrate the possibility.

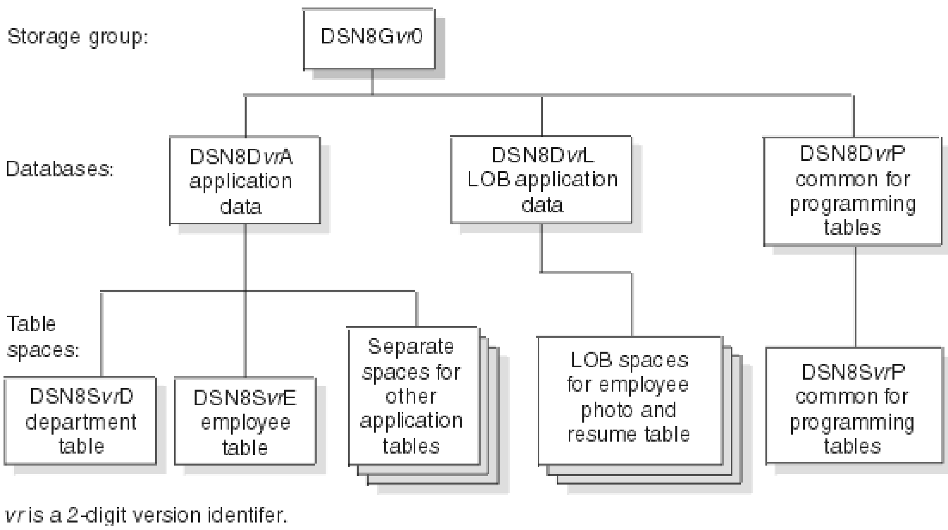


Figure 79. Relationship among sample databases and table spaces

In addition to the storage group and databases that are shown in the preceding figure, the storage group DSN8G11U and database DSN8D11U are created when you run DSNTEJ2A.

Storage group for sample application data

Sample application data is stored in storage group DSN8G110. The default storage group, SYSDEFLT, which is created when Db2 is installed, is not used to store sample application data.

GUPI

The storage group that is used to store sample application data is defined by the following statement:

```
CREATE STOGROUP DSN8G110
  VOLUMES (DSNV01)
  VCAT   DSNC110;
```

GUPI

Databases for sample application data

Sample application data is stored in several different databases. The default database that is created when Db2 is installed is not used to store the sample application data.

GUPI DSN8D11P is the database that is used for tables that are related to programs. The other databases are used for tables that are related to applications. The databases are defined by the following statements:

```
CREATE DATABASE DSN8D11A
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11P
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11L
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11E
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID UNICODE;

CREATE DATABASE DSN8D11U
  STOGROUP DSN8G11U
  CCSID EBCDIC;
```

GUPI

Table spaces for sample application data

The table spaces that are not explicitly defined are created implicitly in the DSN8D11A database, using the default space attributes.

GUPI

The following SQL statements explicitly define a series of table spaces.

```
CREATE TABLESPACE DSN8S11D
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S11E
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  NUMPARTS 4
    (PART 1 USING STOGROUP DSN8G110
                  PRIQTY 12
                  SECQTY 12,
     PART 3 USING STOGROUP DSN8G110
                  PRIQTY 12
                  SECQTY 12)
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  COMPRESS YES
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11B
  IN DSN8D11L
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE
  LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE LOB TABLESPACE DSN8S11M
  IN DSN8D11L
  LOG NO;

CREATE LOB TABLESPACE DSN8S11L
  IN DSN8D11L
  LOG NO;

CREATE LOB TABLESPACE DSN8S11N
  IN DSN8D11L
  LOG NO;
```

```
CREATE TABLESPACE DSN8S11C
  IN DSN8D11P
  USING STOGROUP DSN8G110
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S11P
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE ROW
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11R
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11S
   IN DSN8D11A
   USING STOGROUP DSN8G110
            PRIQTY 20
            SECQTY 20
            ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81Q
   IN DSN8D81P
   USING STOGROUP DSN8G810
            PRIQTY 160
            SECQTY 80
   SEGSIZE 4
   LOCKSIZE PAGE
   BUFFERPOOL BP0
```

```
   CLOSE NO
   CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81U
   IN DSN8D81E
   USING STOGROUP DSN8G810
             PRIQTY 5
             SECQTY 5
             ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID UNICODE;
```

GUPI

SYSDUMMY*x* tables

Db2 for z/OS provides a set of SYSDUMMY*x* catalog tables.

GUPI

The last character of the table name identifies the associated encoding scheme as follows:

- SYSIBM.SYSDUMMY1 uses the EBCDIC encoding scheme.
- SYSIBM.SYSDUMMYE uses the EBCDIC encoding scheme.
- SYSIBM.SYSDUMMYA uses the ASCII encoding scheme.
- SYSIBM.SYSDUMMYU uses the UNICODE encoding scheme.

Although the SYSDUMMY*x* tables are implemented as catalog tables, they are similar to sample tables, and are used in some examples in the Db2 for z/OS documentation.

You can use any of the SYSDUMMY*x* tables when you need to write a query, but no data from a table is referenced. In any query, you must specify a table reference in the FROM clause, but if the query does not reference any data from the table, it does not matter which table is referenced in the FROM clause. Each of the SYSDUMMY*x* tables contains one row, so a SYSDUMMY*x* table can be referenced in a SELECT INTO statement without the need for a predicate to limit the result to a single row of data.

For example, when you want to retrieve the value of a special register or global variable, you can use a query that references a SYSDUMMY*x* table.

```
SELECT CURRENT PATH -- Retrieve the value of a special register
 INTO :myvar
 FROM SYSIBM.SYSDUMMY1;
SELECT CLIENT_IPADDR -- Retrieve the value of a special register
 INTO :myvar
 FROM SYSIBM.SYSDUMMY1;
```

Sometimes when Db2 for z/OS processes an SQL statement, the statement is rewritten, and a reference to a SYSDUMMY*x* table is added. For example, some of the internal rewrites that result in adding a reference to a SYSDUMMY*x* table are for processing the search condition of a trigger, or the SQL PL control statements IF, REPEAT, RETURN, or WHILE. In these situations, the privilege set must include the SELECT privilege on the SYSDUMMY*x* table that is referenced.

GUPI

Related concepts

Objects with different CCSIDs in the same SQL statement (Db2 Internationalization Guide)

Related tasks

Avoiding character conversion for LOB locators

In certain situations, Db2 materializes the entire LOB value and converts it to the encoding scheme of a particular SQL statement. This extra processing can degrade performance and should be avoided.

Related reference

SYSDUMMY1 catalog table (Db2 SQL)

SYSDUMMYA catalog table (Db2 SQL)
SYSDUMMYE catalog table (Db2 SQL)
SYSDUMMYU catalog table (Db2 SQL)

Db2 productivity-aid sample programs

Db2 provides four sample programs that many users find helpful as productivity aids. These programs are shipped as source code, so you can modify them to meet your needs.

DSNTIAUL
DSNTIAUL is a sample program for unloading data, as an alternative to the UNLOAD utility. DSNTIAUL is written in the assembler language. DSNTIAUL can unload some or all rows from up to 100 Db2 tables. With DSNTIAUL, you can unload data of any Db2 built-in data type or distinct type. DSNTIAUL unloads the rows in a form that is compatible with the LOAD utility and generates utility control statements for LOAD. You can also used DSNTIAUL to execute any SQL non-SELECT statement that can be executed dynamically.

DSNTIAD
The DSNTIAD sample program can issue any SQL statement that can be executed dynamically, except for SELECT statements. DSNTIAD is written in assembler language.

DSNTEP2
DSNTEP2 is a sample dynamic SQL program that can issue any SQL statement that can be executed dynamically. DSNTEP2 is written in PL/I language and available in two versions: a source version that you can modify to meet your needs or an object code version that you can use without the need for a PL/I compiler.

DSNTEP4
The DSNTEP4 sample program is identical to DSNTEP2, except that it uses multi-row fetch for increased performance. DSNTEP4 is written in PL/I language and available in two versions: a source version that you can modify to meet your needs or an object code version that you can use without the need for a PL/I compiler.

Because these four programs also accept the static SQL statements CONNECT, SET CONNECTION, and RELEASE, you can use the programs to access Db2 tables at remote locations.

Preparing the productivity-aid sample programs

DSNTIAUL and DSNTIAD are shipped only as source code, so you must precompile, assemble, link, and bind them before you can use them. If you want to use the source code version of DSNTEP2 or DSNTEP4, you must precompile, compile, link, and bind it. You need to bind the object code version of DSNTEP2 or DSNTEP4 before you can use them. Usually a system administrator prepares the programs as part of the installation process. The following table indicates the installation jobs that prepare each sample program. All installation jobs are in data set DSN1110.SDSNSAMP.

Table 176. Jobs that prepare DSNTIAUL, DSNTIAD, DSNTEP2, and DSNTEP4

Program name	Program preparation job
DSNTIAUL	DSNTEJ2A
DSNTIAD	DSNTIJTM
DSNTEP2 (source)	DSNTEJ1P
DSNTEP2 (object)	DSNTEJ1L
DSNTEP4 (source)	DSNTEJ1P
DSNTEP4 (object)	DSNTEJ1L

Running the productivity-aid sample programs

To run the sample programs, use the RUN (DSN) command. For more information, see RUN (DSN) (Db2 Commands) and the following topics.

Retrieval of UTF-16 Unicode data by DSNTEP2, DSNTEP4, and DSNTIAUL

You can use DSNTEP2, DSNTEP4, and DSNTIAUL to retrieve Unicode UTF-16 graphic data. However, these programs might not be able to display some characters, if those characters have no mapping in the target SBCS EBCDIC CCSID.

Related reference

RUN (DSN) (Db2 Commands)

Db2 for z/OS Exchange

DSNTIAUL sample program

You can use the DSNTIAUL program to unload data from Db2 tables into sequential data sets. The data is copied to the data sets and is not deleted from the table.

DSNTIAUL is a sample program for unloading data, as an alternative to the UNLOAD utility. DSNTIAUL is written in the assembler language. DSNTIAUL can unload some or all rows from up to 100 Db2 tables. With DSNTIAUL, you can unload data of any Db2 built-in data type or distinct type. DSNTIAUL unloads the rows in a form that is compatible with the LOAD utility and generates utility control statements for LOAD. You can also used DSNTIAUL to execute any SQL non-SELECT statement that can be executed dynamically.

When multi-row fetch is used, parallelism might be disabled in the last parallel group in the top-level query block for a query. For very simple queries, parallelism might be disabled for the entire query when multi-row fetch is used. To obtain full parallelism when running DSNTIAUL, switch DSNTIAUL to single-row fetch mode by specifying 1 for the *number-of-rows-per-fetch* parameter.

DSNTIAUL uses SQL to access Db2. Operations on a row-level or column-level access control enforced table are subject to the rules specified for the access control. If the table is row-level access control enforced, DSNTIAUL receives and returns only the rows of the table that satisfy the row permissions for the user. If the table is column-level access control enforced, DSNTIAUL receives and returns the values in the column values as modified by the column masks for the user.

Preparing the DSNTIAUL sample program

Before you can use the DSNTIAUL sample program, you must precompile, assemble, link, and bind it first.

For more information, see “Db2 productivity-aid sample programs” on page 996.

Running the DSNTIAUL sample program

To run the DSNTIAUL sample program, use the RUN (DSN) command and specify the following load module and plan name.

Load module name	DSNTIAUL
Plan name	DSNTIBB1

For more about the RUN command, see RUN (DSN) (Db2 Commands).

DSNTIAUL parameters

SQL

Specify SQL to indicate that your input data set contains one or more complete SQL statements, each of which ends with a semicolon. You can include any SQL statement that can be executed dynamically in your input data set. In addition, you can include the static SQL statements CONNECT, SET CONNECTION, or RELEASE. Static SQL statements must be uppercase.

DSNTIAUL uses the SELECT statements to determine which tables to unload and dynamically executes all other statements except CONNECT, SET CONNECTION, and RELEASE. DSNTIAUL executes CONNECT, SET CONNECTION, and RELEASE statically to connect to remote locations.

number-of-rows-per-fetch
Specify a number from 1 to 32767 to specify the number of rows that DSNTIAUL retrieves for each SQL FETCH operation. If you do not specify this number, DSNTIAUL retrieves 100 rows for each FETCH. This parameter can be specified with the SQL parameter.

If the LOBFILE parameter is also specified, and the result set of a FETCH operation can contain NULL LOB values, *number-of-rows-per-fetch* must be 1.

TOLWARN
Specify NO (the default) or YES to indicate whether DSNTIAUL continues to retrieve rows after receiving an SQL warning:

(NO)
If a warning occurs when DSNTIAUL executes an OPEN or FETCH to retrieve rows, DSNTIAUL stops retrieving rows. If the SQLWARN1, SQLWARN2, SQLWARN6, or SQLWARN7 flag is set when DSNTIAUL executes a FETCH to retrieve rows, DSNTIAUL continues to retrieve rows.

(YES)
If a warning occurs when DSNTIAUL executes an OPEN or FETCH to retrieve rows, DSNTIAUL continues to retrieve rows.

(QUIET)
The same as YES except that the program suppresses all SQL warning messages from OPEN or FETCH statements if the SQLCODE is 0 or greater.

LOBFILE(*prefix*)
Specify LOBFILE to indicate that you want DSNTIAUL to dynamically allocate data sets, each to receive the full content of a LOB cell. (A LOB cell is the intersection of a row and a LOB column.) If you do not specify the LOBFILE option, you can unload up to only 32 KB of data from a LOB column.

prefix
Specify a high-level qualifier for these dynamically allocated data sets. You can specify up to 17 characters. The qualifier must conform with the rules for TSO data set names.

DSNTIAUL uses a naming convention for these dynamically allocated data sets of `prefix.Qiiiiiii.Cjjjjjjj.Rkkkkkkk`, where these qualifiers have the following values:

prefix
The high-level qualifier that you specify in the LOBFILE option.

Q*iiiiiii*
The sequence number (starting from 0) of a query that returns one or more LOB columns

C*jjjjjjj*
The sequence number (starting from 0) of a column in a query that returns one or more LOB columns

R*kkkkkkk*
The sequence number (starting from 0) of a row of a result set that has one or more LOB columns.

The generated LOAD statement contains LOB file reference variables that can be used to load data from these dynamically allocated data sets.

If you do not specify the SQL parameter, your input data set must contain one or more single-line statements (without a semicolon) that use the following syntax:

```
table or view name [WHERE conditions] [ORDER BY columns]
```

Each input statement must be a valid SQL SELECT statement with the clause SELECT * FROM omitted and with no ending semicolon. DSNTIAUL generates a SELECT statement for each input statement by appending your input line to SELECT * FROM, then uses the result to determine which tables to unload. For this input format, the text for each table specification can be a maximum of 72 bytes and must not span multiple lines.

You can use the input statements to specify SELECT statements that join two or more tables or select specific columns from a table. If you specify columns, you need to modify the LOAD statement that DSNTIAUL generates.

DSNTIAUL data sets

Data set
Description

SYSIN
Input data set.

If you specify the SQL parameter, you can enter bracketed comments in DSNTIAUL input that includes dynamic SQL statements. Bracketed comments are not supported if the input includes the static SQL statements CONNECT, SET CONNECTION, or RELEASE. Bracketed comments begin with /* and end with */.

The record length for the input data set must be at least 72 bytes. DSNTIAUL reads only the first 72 bytes of each record.

SYSPRINT
Output data set. DSNTIAUL writes informational and error messages in this data set.

The record length for the SYSPRINT data set is 121 bytes.

SYSPUNCH
Output data set. DSNTIAUL writes the LOAD utility control statements in this data set.

SYSREC*nn*
Output data sets. The value *nn* ranges from 00 to 99. You can have a maximum of 100 output data sets for a single execution of DSNTIAUL. Each data set contains the data that is unloaded when DSNTIAUL processes a SELECT statement from the input data set. Therefore, the number of output data sets must match the number of SELECT statements (if you specify parameter SQL) or table specifications in your input data set.

Define all data sets as sequential data sets. You can specify the record length and block size of the SYSPUNCH and SYSREC*nn* data sets. The maximum record length for the SYSPUNCH and SYSREC*nn* data sets is 32760 bytes.

DSNTIAUL return codes

Table 177. DSNTIAUL return codes

Return code	Meaning
0	Successful completion.
4	An SQL statement received a warning code. • If TOLWARN(YES) is specified, and the warning occurred on a FETCH or OPEN during the processing of a SELECT statement, Db2 performs the unload operation. • Otherwise if the SQL statement was a SELECT statement, Db2 did not perform the associated unload operation. If Db2 returns a +394, which indicates that it is using optimization hints, or a +395, which indicates one or more invalid optimization hints, Db2 performs the unload operation.

Table 177. DSNTIAUL return codes (continued)

Return code	Meaning
8	An SQL statement received an error code. If the SQL statement was a SELECT statement, Db2 did not perform the associated unload operation or did not complete it.
12	DSNTIAUL could not open a data set, an SQL statement returned a severe error code (-144, -302, -804, -805, -818, -902, -906, -911, -913, -922, -923, -924, or -927), or an error occurred in the SQL message formatting routine.

Examples

Example: using DSNTIAUL to unload a subset of rows in a table

Suppose that you want to unload the rows for department D01 from the project table. Because you can fit the table specification on one line, and you do not want to execute any non-SELECT statements, you do not need the SQL parameter. Your invocation looks like the one that is shown in the following figure:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIBB1) -
       LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
DSN8B10.PROJ WHERE DEPTNO='D01'
```

Example: using DSNTIAUL to unload rows in more than one table

Suppose that you also want to use DSNTIAUL to perform the following actions:

- Unload all rows from the project table
- Unload only rows from the employee table for employees in departments with department numbers that begin with D, and order the unloaded rows by employee number
- Lock both tables in share mode before you unload them
- Retrieve 250 rows per fetch

For these activities, you must specify the SQL parameter and the *number-of-rows-per-fetch* parameter when you run DSNTIAUL. Your DSNTIAUL invocation is shown in the following figure:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIBB1) PARMS('SQL,250') -
       LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSREC01 DD DSN=DSN8UNLD.SYSREC01,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
LOCK TABLE DSN8B10.EMP IN SHARE MODE;
```

```
LOCK TABLE DSN8B10.PROJ IN SHARE MODE;
SELECT * FROM DSN8B10.PROJ;
SELECT * FROM DSN8B10.EMP
  WHERE WORKDEPT LIKE 'D%'
  ORDER BY EMPNO;
```

Example: using DSNTIAUL to obtain LOAD utility control statements

If you want to obtain the LOAD utility control statements for loading rows into a table, but you do not want to unload the rows, you can set the data set names for the SYSREC*nn* data sets to DUMMY. For example, to obtain the utility control statements for loading rows into the department table, you invoke DSNTIAUL as shown in the following figure:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIBB1) -
       LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DUMMY
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
DSN8B10.DEPT
```

Example: using DSNTIAUL to unload LOB data

This example uses the sample LOB table with the following structure:

```
CREATE TABLE DSN8B10.EMP_PHOTO_RESUME
( EMPNO CHAR(06) NOT NULL,
EMP_ROWID ROWID NOT NULL GENERATED ALWAYS,
PSEG_PHOTO BLOB(500K),
BMP_PHOTO BLOB(100K),
RESUME CLOB(5K),
PRIMARY KEY (EMPNO))
IN DSN8D11L.DSN8S11B
CCSID EBCDIC;
```

The following call to DSNTIAUL unloads the sample LOB table. The parameters for DSNTIAUL indicate the following options:

- The SQL parameter specifies that the input data set (SYSIN) contains SQL.
- The *number-of-rows-per-fetch* parameter value of 1 specifies that DSNTIAUL is to retrieve one row for each FETCH operation. A value of 1 is necessary if the LOB columns that you unload might contain NULL values.
- The LOBFILE parameter value of LOBFILE(DSN8UNLD) specifies that DSNTIAUL places the LOB data in data sets with a high-level qualifier of DSN8UNLD.

```
//UNLOAD   EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB91) -
  PARMS('SQL,1,LOBFILE(DSN8UNLD)') -
  LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSREC00  DD DSN=DSN8UNLD.SYSREC00,
//             UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//             VOL=SER=SCR03,RECFM=FB
//SYSPUNCH  DD DSN=DSN8UNLD.SYSPUNCH,
//             UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//             VOL=SER=SCR03,RECFM=FB
//SYSIN     DD *
  SELECT * FROM DSN8B10.EMP_PHOTO_RESUME;
```

Given that the sample LOB table has 4 rows of data, DSNTIAUL produces the following output:

- Data for columns EMPNO and EMP_ROWID are placed in the data set that is allocated according to the SYSREC00 DD statement. The data set name is DSN8UNLD.SYSREC00
- A generated LOAD statement is placed in the data set that is allocated according to the SYSPUNCH DD statement. The data set name is DSN8UNLD.SYSPUNCH
- The following data sets are dynamically created to store LOB data:
  - DSN8UNLD.Q0000000.C0000002.R0000000
  - DSN8UNLD.Q0000000.C0000002.R0000001
  - DSN8UNLD.Q0000000.C0000002.R0000002
  - DSN8UNLD.Q0000000.C0000002.R0000003
  - DSN8UNLD.Q0000000.C0000003.R0000000
  - DSN8UNLD.Q0000000.C0000003.R0000001
  - DSN8UNLD.Q0000000.C0000003.R0000002
  - DSN8UNLD.Q0000000.C0000003.R0000003
  - DSN8UNLD.Q0000000.C0000004.R0000000
  - DSN8UNLD.Q0000000.C0000004.R0000001
  - DSN8UNLD.Q0000000.C0000004.R0000002
  - DSN8UNLD.Q0000000.C0000004.R0000003

  For example, DSN8UNLD.Q0000000.C0000004.R0000001 means that the data set contains data that is unloaded from the second row (R0000001) and the fifth column (C0000004) of the result set for the first query (Q0000000).

DSNTIAD sample program

You can use the DSNTIAD program to execute dynamic SQL statements other than SELECT statements.

The DSNTIAD sample program can issue any SQL statement that can be executed dynamically, except for SELECT statements. DSNTIAD is written in assembler language.

Preparing the DSNTIAD sample program

Before you can use the DSNTIAD sample program, you must precompile, assemble, link, and bind it first.

For more information, see "Db2 productivity-aid sample programs" on page 996

Running the DSNTIAD sample program

To run the DSNTIAD sample program, use the RUN (DSN) command and specify the following load module and plan name.

Load module name	DSNTIAD
Plan name	DSNTIAB1

For more about the RUN command, see RUN (DSN) (Db2 Commands).

DSNTIAD parameters

RC0

If you specify this parameter, DSNTIAD ends with return code 0, even if the program encounters SQL errors. If you do not specify RC0, DSNTIAD ends with a return code that reflects the severity of the errors that occur. Without RC0, DSNTIAD terminates if more than 10 SQL errors occur during a single execution.

SQLTERM(*termchar*)

Specify this parameter to indicate the character that you use to end each SQL statement. You can use any special character **except** one of those listed in the following table. SQLTERM(;) is the default.

Table 178. Invalid special characters for the SQL terminator

Name	Character	Hexadecimal representation
blank		X'40'
comma	,	X'6B'
double quotation mark	"	X'7F'
left parenthesis	(	X'4D'
right parenthesis	)	X'5D'
single quotation mark	'	X'7D'
underscore	_	X'6D'

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons.

For example, suppose that you specify the parameter SQLTERM(#) to indicate that the character # is the statement terminator. Then a CREATE TRIGGER statement with embedded semicolons looks like this:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

A CREATE PROCEDURE statement with embedded semicolons looks like the following statement:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
  LANGUAGE SQL
  BEGIN
    DECLARE SQLCODE INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET SCODE = SQLCODE;
    UPDATE TBL1 SET COL1 = PARM1;
  END #
```

Be careful to choose a character for the statement terminator that is not used within the statement.

DSNTIAD data sets

Data set
Description

SYSIN

Input data set. In this data set, you can enter any number of non-SELECT SQL statements. Each SQL statement must be terminated with the SQL termination character. If you specify the SQLTERM(*termchar*) parameter, *termchar* is the SQL termination character. Otherwise, the SQL termination character is a semicolon. A statement can span multiple lines, but DSNTIAD reads only the first 72 bytes of each line.

Comments in DSNTIAD input are not supported.

SYSPRINT

Output data set. DSNTIAD writes informational and error messages in this data set. DSNTIAD sets the record length of this data set to 121 bytes and the block size to 1210 bytes.

Define all data sets as sequential data sets.

DSNTIAD return codes

Table 179. DSNTIAD return codes

Return code	Meaning
0	Successful completion, or the user-specified parameter RC0.
4	An SQL statement received a warning code.
8	An SQL statement received an error code.
12	DSNTIAD could not open a data set, the length of an SQL statement was more than 2 MB, an SQL statement returned a severe error code (-8*nn* or -9*nn*), or an error occurred in the SQL message formatting routine.

Example: invoking the DSNTIAD program

Suppose that you want to execute 20 UPDATE statements, and you do not want DSNTIAD to terminate if more than 10 errors occur. Your invocation looks like the one that is shown in the following figure:

```
//RUNTIAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIAB1) PARMS('RC0') -
      LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
UPDATE DSN8B10.PROJ SET DEPTNO='J01' WHERE DEPTNO='A01';
UPDATE DSN8B10.PROJ SET DEPTNO='J02' WHERE DEPTNO='A02';
⋮
UPDATE DSN8B10.PROJ SET DEPTNO='J20' WHERE DEPTNO='A20';
```

DSNTEP2 and DSNTEP4 sample programs

You can use the DSNTEP2 or DSNTEP4 programs to execute SQL statements dynamically.

DSNTEP2 is a sample dynamic SQL program that can issue any SQL statement that can be executed dynamically. DSNTEP2 is written in PL/I language and available in two versions: a source version that you can modify to meet your needs or an object code version that you can use without the need for a PL/I compiler.

The DSNTEP4 sample program is identical to DSNTEP2, except that it uses multi-row fetch for increased performance. DSNTEP4 is written in PL/I language and available in two versions: a source version that you can modify to meet your needs or an object code version that you can use without the need for a PL/I compiler.

When multi-row fetch is used, parallelism might be disabled for the last parallel group in the top-level query block, or entirely disabled for very simple queries. To obtain full parallelism, use DSNTEP2 or specify the control option SET MULT_FETCH 1 for DSNTEP4.

DSNTEP2 and DSNTEP4 write their results to the data set that is defined by the SYSPRINT DD statement. SYSPRINT data must have a logical record length that matches the PAGEWIDTH value in the DSNTEP2 or DSNTEP4 source program. If you use the original version of the program that is shipped with Db2, the logical record length is 133 bytes. If the SYSPRINT data do not have the same logical record length as the PAGEWIDTH value, the program issues return code 12 with abend U4038 and reason code 1. This abend occurs due to the PL/I file exception error IBM0201S ONCODE=81. The following error message is issued:

```
The UNDEFINEDFILE condition was raised because of conflicting DECLARE
and OPEN attributes (FILE= SYSPRINT).
```

If you use applications or other automation to process output from DSNTEP2 or DSNTEP4, be aware that minor changes in the format can occur as a result of service or enhancements. Such changes might require you to adjust your processes that use the output of these programs.

Important: When you allocate a new data set with the SYSPRINT DD statement, either specify a DCB with RECFM=FBA and LRECL=*nnn*, where *nnn* matches the PAGEWIDTH value in the source program, or do not specify the DCB parameter.

Preparing the DSNTEP2 and DSTEP4 sample programs

Before you can use the DSNTEP2 and DSTEP4 sample programs, you must prepare them. If you use the source-code versions of DSNTEP2 or DSNTEP4, you must precompile, compile, link, and bind them first. If you use the object-code versions of DSNTEP2 or DSNTEP4 you must bind them first.

For more information, see "Preparing the productivity-aid sample programs" in "Db2 productivity-aid sample programs" on page 996.

Running the DSNTEP2 and DSNTEP4 sample programs

To run the DSNTEP2 and DSNTEP4 sample programs, use the RUN (DSN) command and specify the following load module and plan name.

DSNTEP2 load module and plan

Load module	DSNTEP2
Plan	DSNTEPB1

DSNTEP4 load module and plan

Load module	DSNTEP4
Plan	DSNTP411

For more about the RUN command, see RUN (DSN) (Db2 Commands).

DSNTEP2 and DSNTEP4 parameters

ALIGN(MID) or ALIGN(LHS)
Specifies the alignment.

ALIGN(MID)
Specifies that DSNTEP2 or DSNTEP4 output should be centered. `ALIGN(MID)` is the default.

ALIGN(LHS)
Specifies that the DSNTEP2 or DSNTEP4 output should be left-justified.

NOMIXED or MIXED
Specifies whether DSNTEP2 or DSNTEP4 contains any DBCS characters.

NOMIXED
Specifies that the DSNTEP2 or DSNTEP4 input contains no DBCS characters. `NOMIXED` is the default.

MIXED
Specifies that the DSNTEP2 or DSNTEP4 input contains some DBCS characters.

PREPWARN
Specifies that DSNTEP2 or DSNTEP4 is to display details about any SQL warnings that are encountered at PREPARE time.

Regardless of whether you specify PREPWARN, when an SQL warning is encountered at PREPARE time, the program displays the message `SQLWARNING ON PREPARE` and sets the return code to 4. If DSNTEP2 is bound with REOPT(ALWAYS), it is normal that an SQL warning is encountered before PREPARE and return code is set to 4 if incremental bind happens. When you specify PREPWARN, the program also displays the details about any SQL warnings.

SQLFORMAT
Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements before passing them to Db2. Select one of the following options:

SQL
This is the preferred mode for SQL statements other than SQL procedural language. When you use this option, which is the default, DSNTEP2 or DSNTEP4 collapses each line of an SQL statement into a single line before passing the statement to Db2. DSNTEP2 or DSNTEP4 also discards all SQL comments.

SQLCOMNT
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, behavior is similar to SQL mode, except that DSNTEP2 or DSNTEP4 does not discard SQL comments. Instead, it automatically terminates each SQL comment with a line feed character (hex 25), unless the comment is already terminated by one or more line formatting characters. Use this option to process SQL procedural language with minimal modification by DSNTEP2 or DSNTEP4.

SQLPL
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, DSNTEP2 or DSNTEP4 retains SQL comments and terminates each line of an SQL statement with a line feed character (hex 25) before passing the statement to Db2. Lines that end with a split token are not terminated with a line feed character. Use this mode to obtain improved diagnostics and debugging of SQL procedural language.

SQLTERM(*termchar*)
Specifies the character that you use to end each SQL statement. You can use any character except one of those that are listed in Table 178 on page 1003. `SQLTERM(;)` is the default.

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons.

See "Example: changing the SQL terminator" on page 1009.

TOLWARN
Indicates whether DSNTEP2 or DSNTEP4 continues to process SQL SELECT statements after receiving an SQL warning. You can specify one of the following values:

NO
Indicates that the program stops processing the SELECT statement if a warning occurs when the program executes an OPEN or FETCH for a SELECT statement. NO is the default value for TOLWARN.

The following exceptions exist:

- If SQLCODE +445 or SQLCODE +595 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement.
- If SQLCODE +354 occurs when DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement.
- If SQLCODE +802 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement if the TOLARTHWRN control statement is set to YES.

YES
Indicates that the program continues to process the SELECT statement if a warning occurs when the program executes an OPEN or FETCH for a SELECT statement.

QUIET
The same as YES except that the program suppresses all SQL warning messages from OPEN or FETCH statements if the SQLCODE is 0 or greater.

Settings that you can change in the DSNTEP2 or DSNTEP4 source programs to modify output formatting

If the DSNTEP2 or DSNTEP4 output formatting does not meet your requirements, you can change some variables in the source code to modify the formatting.

For example, if you want to increase the maximum width of a page, the maximum size of a print line, and the maximum number of characters in the output of a character column, you can increase the values of the PAGEWIDTH, MAXPAGWD, and MAXCOLWD variable values.

For a complete description of the settings that you can change, see the information under PROGRAM SIZES in the DSNTEP2 and DSNTEP4 prologs.

DSNTEP2 and DSNTEP4 data sets

The following data sets are used by DSNTEP2 and DSNTEP4:

SYSIN

Input data set. In this data set, you can enter any number of SQL statements, each terminated with a semicolon. A statement can span multiple lines, but DSNTEP2 or DSNTEP4 reads only the first 72 bytes of each line. You must explicitly commit any SQL statements except the last one.

You can enter comments in DSNTEP2 or DSNTEP4 input with an asterisk (*) in column 1 or two hyphens (--) anywhere on a line. Text that follows the asterisk is considered to be comment text. Text that follows two hyphens can be comment text or a control statement.Comments are not considered in dynamic statement caching. Comments and control statements cannot span lines.

You can enter control statements of the following form in the DSNTEP2 and DSNTEP4 input data set:

```
--#SET control-option value
```

You can specify the following control options. If you specify a value of NO for any of the options in this list, the program behaves as if you did not specify the parameter.

TERMINATOR

The SQL statement terminator. *value* is any single-byte character other than one of those that are listed in "DSNTIAD sample program" on page 1002. The default is the value of the SQLTERM parameter.

See "Example: changing the SQL terminator within a series of SQL statements" on page 1010.

ROWS_FETCH

The number of rows that are to be fetched from the result table. *value* is a numeric literal between -1 and the number of rows in the result table. -1 means that all rows are to be fetched. The default is -1.

ROWS_OUT

The number of fetched rows that are to be sent to the output data set. *value* is a numeric literal between -1 and the number of fetched rows. -1 means that all fetched rows are to be sent to the output data set. The default is -1.

MULT_FETCH

This option is valid only for DSNTEP4. Use MULT_FETCH to specify the number of rows that are to be fetched at one time from the result table. The default fetch amount for DSNTEP4 is 100 rows, but you can specify from 1 to 32676 rows.

TOLWARN

Indicates whether DSNTEP2 or DSNTEP4 continues to process SQL SELECT statements after receiving an SQL warning. You can specify one of the following values:

NO

Indicates that the program stops processing the SELECT statement if a warning occurs when the program executes an OPEN or FETCH for a SELECT statement. NO is the default value for TOLWARN.

The following exceptions exist:

- If SQLCODE +445 or SQLCODE +595 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement.
- If SQLCODE +354 occurs when DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement.
- If SQLCODE +802 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, the program continues to process the SELECT statement if the TOLARTHWRN control statement is set to YES.

YES
Indicates that the program continues to process the SELECT statement if a warning occurs when the program executes an OPEN or FETCH for a SELECT statement.

TOLARTHWRN
Indicates whether DSNTEP2 and DSNTEP4 continue to process an SQL SELECT statement after an arithmetic SQL warning (SQLCODE +802) is returned. *value* is either NO (the default) or YES.

PREPWARN
Specifies that DSNTEP2 or DSNTEP4 is to display details about any SQL warnings that are encountered at PREPARE time.

Regardless of whether you specify PREPWARN, when an SQL warning is encountered at PREPARE time, the program displays the message `SQLWARNING ON PREPARE` and sets the return code to 4. If DSNTEP2 is bound with REOPT(ALWAYS), it is normal that an SQL warning is encountered before PREPARE and return code is set to 4 if incremental bind happens. When you specify PREPWARN, the program also displays the details about any SQL warnings.

SQLFORMAT
Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements before passing them to Db2. Select one of the following options:

SQL
This is the preferred mode for SQL statements other than SQL procedural language. When you use this option, which is the default, DSNTEP2 or DSNTEP4 collapses each line of an SQL statement into a single line before passing the statement to Db2. DSNTEP2 or DSNTEP4 also discards all SQL comments.

SQLCOMNT
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, behavior is similar to SQL mode, except that DSNTEP2 or DSNTEP4 does not discard SQL comments. Instead, it automatically terminates each SQL comment with a line feed character (hex 25), unless the comment is already terminated by one or more line formatting characters. Use this option to process SQL procedural language with minimal modification by DSNTEP2 or DSNTEP4.

SQLPL
This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, DSNTEP2 or DSNTEP4 retains SQL comments and terminates each line of an SQL statement with a line feed character (hex 25) before passing the statement to Db2. Lines that end with a split token are not terminated with a line feed character. Use this mode to obtain improved diagnostics and debugging of SQL procedural language.

MAXERRORS
Specifies that number of errors that DSNTEP2 and DSNTEP4 handle before processing stops. The default is 10. Use a value of -1 to indicate that a program is to tolerate an unlimited number of errors.

SYSPRINT
Output data set. DSNTEP2 and DSNTEP4 write informational and error messages in this data set. DSNTEP2 and DSNTEP4 write output records of no more than 133 bytes.

Define all data sets as sequential data sets.

DSNTEP2 and DSNTEP4 return codes

Table 180. DSNTEP2 and DSNTEP4 return codes

Return code	Meaning
0	Successful completion.
4	An SQL statement received a warning code.
8	An SQL statement received an error code.
12	The length of an SQL statement was more than 2097152 bytes, an SQL statement returned a severe error code (-8*nn* or -9*nn*), or an error occurred in the SQL message formatting routine.

Examples

Example: invoking DSNTEP2

Suppose that you want to use DSNTEP2 to execute SQL SELECT statements that might contain DBCS characters. You also want left-aligned output. Your invocation looks like the following example.

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEPB1) PARMS('/ALIGN(LHS) MIXED TOLWARN(YES)') -
       LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
SELECT * FROM DSN8B10.PROJ;
```

Example invoking DSNTEP4

Suppose that you want to use DSNTEP4 to execute SQL SELECT statements that might contain DBCS characters, and you want center-aligned output. You also want DSNTEP4 to fetch 250 rows at a time. Your invocation looks like the one in the following figure:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTEP4) PLAN(DSNTEPB1) PARMS('/ALIGN(MID) MIXED') -
       LIB('DSN1110.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
--#SET MULT_FETCH 250
SELECT * FROM DSN8B10.EMP;
```

Example: changing the SQL terminator

Suppose that you specify the parameter SQLTERM(#) to indicate that the character # is the statement terminator. Then a CREATE TRIGGER statement with embedded semicolons looks like this:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

A CREATE PROCEDURE statement with embedded semicolons looks like the following statement:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
  LANGUAGE SQL
  BEGIN
    DECLARE SQLCODE INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET SCODE = SQLCODE;
```

```
    UPDATE TBL1 SET COL1 = PARM1;
  END #
```

Be careful to choose a character for the statement terminator that is not used within the statement.

Example: changing the SQL terminator within a series of SQL statements

You can use the --#SET TERMINATOR control statement. Suppose that you have an existing set of SQL statements to which you want to add a CREATE TRIGGER statement that has embedded semicolons. You can use the default SQLTERM value, which is a semicolon, for all of the existing SQL statements.

Before you execute the CREATE TRIGGER statement, include the `--#SET TERMINATOR #` control statement to change the SQL terminator to the character #:

```
SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJACT;
SELECT * FROM PROJ;
SELECT * FROM PROJACT;
--#SET TERMINATOR #
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

See the following discussion of the SYSIN data set for more information about the --#SET control statement.

Sample applications supplied with Db2 for z/OS

Db2 provides sample applications to help you with Db2 programming techniques and coding practices within each of the four environments: batch, TSO, IMS, and CICS. The sample applications contain various applications that might apply to managing a company.

This topic describes the Db2 sample applications and the environments under which each application runs. It also provides information on how to use the applications, and how to print the application listings.

You can examine the source code for the sample application programs in the online sample library included with the Db2 product. The name of this sample library is DSN1110.SDSNSAMP.

Using the sample applications

You can use the applications interactively by accessing data in the sample tables on screen displays (panels). You can also access the sample tables in batch when using the phone applications. All sample objects have PUBLIC authorization, which makes the samples easier to run.

Related concepts

Db2 programming samples (Db2 Programming samples)

Related reference

Db2 for z/OS Exchange

Types of sample applications

Db2 provides a number of sample applications that manage sample company information. These applications also demonstrate how to use stored procedures, user-defined functions, and LOBs.

Organization application:

The organization application manages the following company information:

- Department administrative structure
- Individual departments
- Individual employees.

Management of information about department administrative structures involves how departments relate to other departments. You can view or change the organizational structure of an individual department, and the information about individual employees in any department. The organization application runs interactively in the ISPF/TSO, IMS, and CICS environments and is available in PL/I and COBOL.

Project application:

The project application manages information about a company's project activities, including the following:

- Project structures
- Project activity listings
- Individual project processing
- Individual project activity estimate processing
- Individual project staffing processing.

Each department works on projects that contain sets of related activities. Information available about these activities includes staffing assignments, completion-time estimates for the project as a whole, and individual activities within a project. The project application runs interactively in IMS and CICS and is available in PL/I only.

Phone application:

The phone application lets you view or update individual employee phone numbers. There are different versions of the application for ISPF/TSO, CICS, IMS, and batch:

- ISPF/TSO applications use COBOL and PL/I.
- CICS and IMS applications use PL/I.
- Batch applications use C, C++, COBOL, FORTRAN, and PL/I.

Stored procedure applications:

There are three sets of stored procedure applications:

IFI applications
These applications let you pass Db2 commands from a client program to a stored procedure, which runs the commands at a Db2 server using the instrumentation facility interface (IFI). There are two sets of client programs and stored procedures. One set has a PL/I client and stored procedure; the other set has a C client and stored procedure.

ODBA application
This application demonstrates how you can use the IMS ODBA interface to access IMS databases from stored procedures. The stored procedure accesses the IMS sample DL/I database. The client program and the stored procedure are written in COBOL.

Utilities stored procedure application
This application demonstrates how to call the utilities stored procedure.

SQL procedure applications

Sample applications are available for both external SQL procedures and native SQL procedures:

- The applications for external SQL procedures demonstrate how to write, prepare, and invoke such procedures. One set of applications demonstrates how to prepare SQL procedures using JCL. The other set of applications shows how to prepare SQL procedures using the SQL procedure processor. The client programs are written in C.
- The sample job for a native SQL procedure shows how to prepare a native SQL procedure, how to manage versions of native SQL procedures, and optionally, how to deploy a native SQL procedure to a remote server. The sample also prepares and executes a sample caller in the C language

WLM refresh application
: This application is a client program that calls the Db2–supplied stored procedure WLM_REFRESH to refresh a WLM environment. This program is written in C.

System parameter reporting application
: This application is a client program that calls the Db2–supplied stored procedure ADMIN_INFO_SYSPARM to display the current settings of system parameters. This program is written in C.

All stored procedure applications run in the TSO batch environment.

User-defined function applications:

The user-defined function applications consist of a client program that invokes the sample user-defined functions and a set of user-defined functions that perform the following functions:

- Convert the current date to a user-specified format
- Convert a date from one format to another
- Convert the current time to a user-specified format
- Convert a date from one format to another
- Return the day of the week for a user-specified date
- Return the month for a user-specified date
- Format a floating point number as a currency value
- Return the table name for a table, view, or alias
- Return the qualifier for a table, view or alias
- Return the location for a table, view or alias
- Return a table of weather information

All programs are written in C or C++ and run in the TSO batch environment.

LOB application:

The LOB application demonstrates how to perform the following tasks:

- Define Db2 objects to hold LOB data
- Populate Db2 tables with LOB data using the LOAD utility, or using INSERT and UPDATE statements when the data is too large for use with the LOAD utility
- Manipulate the LOB data using LOB locators

The programs that create and populate the LOB objects use DSNTIAD and run in the TSO batch environment. The program that manipulates the LOB data is written in C and runs under ISPF/TSO.

Application languages and environments for the sample applications

The sample applications demonstrate how to run Db2 applications in the TSO, IMS, or CICS environments.

The following table shows the environments under which each application runs, and the languages the applications use for each environment.

Table 181. Application languages and environments

Programs	ISPF/TSO	IMS	CICS	Batch	SPUFI
Dynamic SQL programs				Assembler PL/I	
Exit routines	Assembler	Assembler	Assembler	Assembler	Assembler

Table 181. Application languages and environments (continued)

Programs	ISPF/TSO	IMS	CICS	Batch	SPUFI
Organization	COBOL	COBOL PL/I	COBOL PL/I		
Phone	COBOL PL/I Assembler[1]	PL/I	PL/I	COBOL FORTRAN PL/I C C++	
Project		PL/I	PL/I		
SQLCA formatting routines		Assembler	Assembler	Assembler	Assembler
Stored procedures		COBOL		PL/I C SQL	
User-defined functions				C C++	
LOBs	C				

Notes:

1. Assembler subroutine DSN8CA.

Sample applications in TSO

A set of Db2 sample applications run in the TSO environment.

Table 182. Sample Db2 applications for TSO

Application	Program name	Preparation JCL member name	Attachment facility	Description
Phone	DSN8BC3	DSNTEJ2C	DSNELI	This COBOL batch program lists employee telephone numbers and updates them if requested.
Phone	DSN8BD3	DSNTEJ2D	DSNELI	This C batch program lists employee telephone numbers and updates them if requested.
Phone	DSN8BE3	DSNTEJ2E	DSNELI	This C++ batch program lists employee telephone numbers and updates them if requested.
Phone	DSN8BP3	DSNTEJ2P	DSNELI	This PL/I batch program lists employee telephone numbers and updates them if requested.
Phone	DSN8BF3	DSNTEJ2F	DSNELI	This FORTRAN program lists employee telephone numbers and updates them if requested.

Table 182. Sample Db2 applications for TSO (continued)

Application	Program name	Preparation JCL member name	Attachment facility	Description
Organization	DSN8HC3	DSNTEJ3C or DSNTEJ6	DSNALI	This COBOL ISPF program displays and updates information about a local department. It can also display and update information about an employee at a local or remote location.
Phone	DSN8SC3	DSNTEJ3C	DSNALI	This COBOL ISPF program lists employee telephone numbers and updates them if requested.
Phone	DSN8SP3	DSNTEJ3P	DSNALI	This PL/I ISPF program lists employee telephone numbers and updates them if requested.
UNLOAD	DSNTIAUL	DSNTEJ2A	DSNELI	This assembler language program unloads the data from a table or view and to produce LOAD utility control statements for the data.
Dynamic SQL	DSNTIAD	DSNTIJTM	DSNELI	This assembler language program dynamically executes non-SELECT statements read in from SYSIN; that is, it uses dynamic SQL to execute non-SELECT SQL statements.
Dynamic SQL	DSNTEP2	DSNTEJ1P or DSNTEJ1L	DSNELI	This PL/I program dynamically executes SQL statements read in from SYSIN. Unlike DSNTIAD, this application can also execute SELECT statements.
Stored procedures[1]	DSN8EP1	DSNTEJ6P	DSNELI	The jobs DSNTEJ6P and DSNTEJ6S prepare a PL/I version of the application. This sample executes Db2 commands using the instrumentation facility interface (IFI).
Stored procedure[1]	DSN8EP2	DSNTEJ6S	DSNRLI	
Stored procedures[1]	DSN8EPU	DSNTEJ6U	DSNELI	The sample that is prepared by job DSNTEJ6U invokes the utilities stored procedure.
Stored procedures[1]	DSN8ED1	DSNTEJ6D	DSNELI	The jobs DSNTEJ6D and DSNTEJ6T prepare a C version of the application. The C stored procedure uses result sets to return commands to the client. This sample executes Db2 commands using the instrumentation facility interface (IFI).
Stored procedures[1]	DSN8ED2	DSNTEJ6T	DSNRLI	
Stored procedures[1]	DSN8EC1	DSNTEJ61	DSNRLI	The sample that is prepared by jobs DSNTEJ61 and DSNTEJ62 demonstrates a stored procedure that accesses IMS databases through the ODBA interface.
Stored procedures[1]	DSN8EC2	DSNTEJ62	DSNELI	
Stored procedures[1]	DSN8ES1	DSNTEJ63	DSNRLI	The sample that is prepared by jobs DSNTEJ63 and DSNTEJ64 demonstrates how to prepare an SQL procedure using JCL.
Stored procedures[1]	DSN8ED3	DSNTEJ64	DSNELI	

Table 182. Sample Db2 applications for TSO (continued)

Application	Program name	Preparation JCL member name	Attachment facility	Description
Stored procedures[1]	DSN8ES2	DSNTEJ65	DSNRLI	The sample that is prepared by job DSNTEJ65 demonstrates how to prepare an SQL procedure using the SQL procedure processor.
Stored procedures[1]	DSN8ED6	DSNTEJ6W	DSNELI	The sample that is prepared by job DSNTEJ6W demonstrates how to prepare and run a client program that calls a Db2–supplied stored procedure to refresh a WLM environment.
Stored procedures[1]	DSN8ED7	DSNTEJ6Z	DSNELI	The sample that is prepared by job DSNTEJ6Z demonstrates how to prepare and run a client program that calls a Db2–supplied stored procedure to display the current settings of system parameters.
Stored procedures[1]	DSN8ED9	DSNTEJ66	DSNELI	The sample that is prepared by job DSNTEJ66 demonstrates how to prepare and run a client program that calls a native SQL procedure, manages versions of that procedure, and optionally, deploys that procedure to a remote server. DSN8ES3 is the sample native SQL procedure and DSN8ED9 is the sample C language caller of DSN8ES3.
Stored procedures[1]	DSN8ES3	DSNTEJ66	not applicable	The sample that is prepared by job DSNTEJ66 demonstrates how to prepare and run a client program that calls a native SQL procedure, manages versions of that procedure, and optionally, deploys that procedure to a remote server. DSN8ES3 is the sample native SQL procedure and DSN8ED9 is the sample C language caller of DSN8ES3.
User-defined functions	DSN8DUAD	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUAT	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUCD	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUCT	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUCY	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUTI	DSNTEJ2U	DSNRLI	These C applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8DUWC	DSNTEJ2U	DSNRLI	The user-defined table function DSN8DUWF can be invoked by the C client program DSN8DUWC.
User-defined functions	DSN8DUWF	DSNTEJ2U	DSNRLI	The user-defined table function DSN8DUWF can be invoked by the C client program DSN8DUWC.
User-defined functions	DSN8EUDN	DSNTEJ2U	DSNRLI	These C++ applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8EUMN	DSNTEJ2U	DSNRLI	These C++ applications consist of a set of user-defined scalar functions that can be invoked through SPUFI or DSNTEP2.

Table 182. Sample Db2 applications for TSO (continued)

Application	Program name	Preparation JCL member name	Attachment facility	Description
User-defined functions	DSN8HDFS	DSNTEJBI	DSNRLI	The user-defined table function HDFS_READ, which is prepared by DSNTEJBI, reads data from a delimiter-separated file in the Hadoop Distributed File System (HDFS). This user-defined function can be invoked through SPUFI or DSNTEP2.
User-defined functions	DSN8JAQL	DSNTEJBI	DSNRLI	The user-defined scalar function JAQL_SUBMIT, which is prepared by DSNTEJBI, invokes an IBM InfoSphere BigInsights Jaql query. This user-defined function can be invoked through SPUFI or DSNTEP2.
LOBs	DSN8DLPL	DSNTEJ71	DSNELI	These applications demonstrate how to populate a LOB column that is greater than 32 KB, manipulate the data using the POSSTR and SUBSTR built-in functions, and display the data in ISPF using GDDM.
LOBs	DSN8DLCT	DSNTEJ71	DSNELI	
LOBs	DSN8DLRV	DSNTEJ73	DSNELI	
LOBs	DSN8DLPV	DSNTEJ75	DSNELI	

Note:

1. All of the stored procedure applications consist of a calling program, a stored procedure program, or both.

Related reference

Data sets that the precompiler uses
When you invoke the precompiler you need to provide data sets that contain input for the precompiler, such as the host programming statements and SQL statements. You also need to provide data sets where the precompiler can store its output, such as the modified source code and diagnostics messages.

DSN8BC3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
        IDENTIFICATION DIVISION.
       *-----------------------
        PROGRAM-ID.    DSN8BC3.

       ****** DSN8BC3 - DB2 SAMPLE PHONE APPLICATION - COBOL - BATCH ***
       *                                                                *
       *   MODULE NAME = DSN8BC3                                        *
       *                                                                *
       *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                   *
       *                       PHONE APPLICATION                        *
       *                       BATCH                                    *
       *                       COBOL                                    *
       *                                                                *
       *LICENSED MATERIALS - PROPERTY OF IBM                            *
       *5605-DB2                                                        *
       *(C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.        *
       *                                                                *
       *STATUS = VERSION 10                                             *
       *                                                                *
       *   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND      *
       *              UPDATES THEM IF DESIRED.                          *
       *                                                                *
       *   NOTES = NONE                                                 *
       *                                                                *
       *   MODULE TYPE = COBOL PROGRAM                                  *
       *      PROCESSOR   = DB2  PRECOMPILER, VS COBOL                  *
```

```
*      MODULE SIZE = SEE LINK EDIT                              *
*      ATTRIBUTES  = NOT REENTRANT OR REUSABLE                  *
*                                                               *
*                                                               *
*   ENTRY POINT = DSN8BC3                                       *
*      PURPOSE = SEE FUNCTION                                   *
*      LINKAGE = INVOKED FROM DSN RUN                           *
*      INPUT   =                                                *
*                                                               *
*                SYMBOLIC LABEL/NAME =  CARDIN                  *
*                DESCRIPTION         =  INPUT REQUEST FILE      *
*                                                               *
*                SYMBOLIC LABEL/NAME =  VPHONE                  *
*                DESCRIPTION         =  VIEW OF TELEPHONE       *
*                                       INFORMATION             *
*                                                               *
*                                                               *
*      OUTPUT  =                                                *
*                                                               *
*                SYMBOLIC LABEL/NAME =  REPORT                  *
*                DESCRIPTION         =  REPORT OF EMPLOYEE      *
*                                       PHONE NUMBERS           *
*                                                               *
*                SYMBOLIC LABEL/NAME =  VEMPLP                  *
*                DESCRIPTION         =  VIEW OF EMPLOYEE        *
*                                       INFORMATION             *
*                                                               *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION               *
*                                                               *
*   EXIT-ERROR =                                                *
*                                                               *
*      RETURN CODE    =  NONE                                   *
*                                                               *
*      ABEND CODES    =  NONE                                   *
*                                                               *
*      ERROR-MESSAGES =                                         *
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED           *
*            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE*
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE              *
*            DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED*
*            DSN8060E - SQL ERROR, RETURN CODE IS:              *
*            DSN8061E - ROLLBACK FAILED, RETURN CODE IS:        *
*            DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'   *
*            DSN8075E - MESSAGE FORMAT ROUTINE ERROR,           *
*                        RETURN CODE IS:                        *
*                                                               *
*   EXTERNAL REFERENCES =                                       *
*      ROUTINES/SERVICES =                                      *
*            DSNTIAR  -     TRANSLATE SQLCA INTO MESSAGES       *
*            DSN8MCG -      ERROR MESSAGE ROUTINE               *
*                                                               *
*      DATA-AREAS        =  NONE                                *
*                                                               *
*      CONTROL-BLOCKS    =                                      *
*            SQLCA    -     SQL COMMUNICATION AREA              *
*                                                               *
*   TABLES =  NONE                                              *
*                                                               *
*                                                               *
*   CHANGE-ACTIVITY = NONE                                      *
*                                                               *
*                                                               *
*  *PSEUDOCODE*                                                 *
*                                                               *
*  PROCEDURE                                                    *
*    GET FIRST INPUT                                            *
*    DO WHILE MORE INPUT                                        *
*      CREATE REPORT HEADING                                    *
*                                                               *
*      CASE (ACTION)                                            *
*                                                               *
*        SUBCASE ('L')                                          *
*          IF LASTNAME IS '*' THEN                              *
*            LIST ALL EMPLOYEES                                 *
*          ELSE                                                 *
*            IF LASTNAME CONTAINS '%' THEN                      *
*              LIST EMPLOYEES GENERIC                           *
*            ELSE                                               *
*              LIST EMPLOYEES SPECIFIC                          *
*        ENDSUB                                                 *
*                                                               *
*        SUBCASE ('U')                                          *
```

```
*          UPDATE PHONENUMBER FOR EMPLOYEE                      *
*          WRITE CONFIRMATION MESSAGE                           *
*        OTHERWISE                                              *
*          INVALID REQUEST                                      *
*        ENDSUB                                                 *
*                                                               *
*      ENDCASE                                                  *
*      GET NEXT INPUT                                           *
*    END                                                        *
*                                                               *
*    IF SQL ERROR OCCURS THEN                                   *
*       DO                                                      *
*        FORMAT ERROR MESSAGE                                   *
*        ROLLBACK                                               *
*       END                                                     *
*  END.                                                         *
*---------------------------------------------------------------*


/
 ENVIRONMENT DIVISION.
*--------------------
 CONFIGURATION SECTION.
 SPECIAL-NAMES.      C01 IS TO-TOP-OF-PAGE.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT CARDIN
            ASSIGN TO DA-S-CARDIN.
     SELECT REPOUT
            ASSIGN TO UT-S-REPORT.

 DATA DIVISION.
*-------------
 FILE SECTION.
 FD      CARDIN
         RECORD CONTAINS 80 CHARACTERS
         BLOCK CONTAINS 0 RECORDS
         LABEL RECORDS ARE OMITTED.
 01  CARDREC                   PIC X(80).

 FD  REPOUT
         RECORD CONTAINS 120 CHARACTERS
         LABEL RECORDS ARE OMITTED
         DATA RECORD IS REPREC.
 01  REPREC                    PIC X(120).
/
 WORKING-STORAGE SECTION.

*****************************************************
* STRUCTURE FOR INPUT                               *
*****************************************************
 01  IOAREA.
         02  ACTION            PIC X(01).
         02  LNAME             PIC X(15).
         02  FNAME             PIC X(12).
         02  ENO               PIC X(06).
         02  NEWNO             PIC X(04).
         02  FILLER            PIC X(42).

*****************************************************
* REPORT HEADER STRUCTURE                           *
*****************************************************
 01  REPHDR1.
         02  FILLER PIC X(29)
             VALUE '----------------------------'.
         02  FILLER PIC X(21)
             VALUE ' TELEPHONE DIRECTORY '.
         02  FILLER PIC X(29)
             VALUE '----------------------------'.
 01  REPHDR2.
         02  FILLER PIC X(09) VALUE 'LAST NAME'.
         02  FILLER PIC X(07) VALUE SPACES.
         02  FILLER PIC X(10) VALUE 'FIRST NAME'.
         02  FILLER PIC X(03) VALUE SPACES.
         02  FILLER PIC X(08) VALUE 'INITIAL'.
         02  FILLER PIC X(07) VALUE 'PHONE'.
         02  FILLER PIC X(09) VALUE 'EMPLOYEE'.
         02  FILLER PIC X(05) VALUE 'WORK'.
         02  FILLER PIC X(04) VALUE 'WORK'.
 01  REPHDR3.
         02  FILLER PIC X(37) VALUE SPACES.
         02  FILLER PIC X(07) VALUE 'NUMBER'.
```

```
         02  FILLER PIC X(09) VALUE 'NUMBER'.
         02  FILLER PIC X(05) VALUE 'DEPT'.
         02  FILLER PIC X(05) VALUE 'DEPT'.
         02  FILLER PIC X(04) VALUE 'NAME'.

*****************************************************
* REPORT STRUCTURE                                  *
*****************************************************
 01  REPDATA.
         02  RLNAME    PIC X(15).
         02  FILLER    PIC X(01) VALUE SPACES.
         02  RFNAME    PIC X(12).
         02  FILLER    PIC X(04) VALUE SPACES.
         02  RMIDINIT  PIC X(01).
         02  FILLER    PIC X(04) VALUE SPACES.
         02  RPHONE    PIC X(04).
         02  FILLER    PIC X(03) VALUE SPACES.
         02  REMPNO    PIC X(06).
         02  FILLER    PIC X(03) VALUE SPACES.
         02  RDEPTNO   PIC X(03).
         02  FILLER    PIC X(02) VALUE SPACES.
         02  RDEPTNAME PIC X(36).

*****************************************************
* WORKAREAS                                         *
*****************************************************
 01  LNAME-WORK.
      49 LNAME-WORKL     PIC S9(4)  COMP.
      49 LNAME-WORKC     PIC X(15).
 01  FNAME-WORK.
      49 FNAME-WORKL     PIC S9(4)  COMP.
      49 FNAME-WORKC     PIC X(12).
 77  INPUT-SWITCH        PIC X          VALUE  'Y'.
         88  NOMORE-INPUT               VALUE  'N'.
 77  NOT-FOUND           PIC S9(9) COMP VALUE  +100.

*****************************************************
* VARIABLES FOR ERROR-HANDLING                      *
*****************************************************
 01  ERROR-MESSAGE.
         02  ERROR-LEN   PIC S9(4)  COMP VALUE +960.
         02  ERROR-TEXT  PIC X(120) OCCURS 10 TIMES
                                    INDEXED BY ERROR-INDEX.
 77  ERROR-TEXT-LEN      PIC S9(9)  COMP VALUE +120.

/****************************************************
* SQL INCLUDE FOR SQLCA                             *
*****************************************************
          EXEC SQL INCLUDE SQLCA  END-EXEC.

*****************************************************
* SQL DECLARATION FOR VIEW VPHONE                   *
*****************************************************
          EXEC SQL DECLARE VPHONE TABLE
                  (LASTNAME       VARCHAR(15)  NOT NULL,
                   FIRSTNAME      VARCHAR(12)  NOT NULL,
                   MIDDLEINITIAL     CHAR(01)  NOT NULL,
                   PHONENUMBER       CHAR(04)          ,
                   EMPLOYEENUMBER    CHAR(06)  NOT NULL,
                   DEPTNUMBER        CHAR(03)  NOT NULL,
                   DEPTNAME       VARCHAR(36)  NOT NULL)
                END-EXEC.

*****************************************************
* STRUCTURE FOR PPHONE RECORD                       *
*****************************************************
 01  PPHONE.
        02  LASTNAME.
              49  LASTNAMEL      PIC S9(4)  COMP.
              49  LASTNAMEC      PIC X(15)  VALUE SPACES.
        02  FIRSTNAME.
              49  FIRSTNAMEL     PIC S9(4)  COMP.
              49  FIRSTNAMEC     PIC X(12)  VALUE SPACES.
        02  MIDDLEINITIAL        PIC X(01).
        02  PHONENUMBER          PIC X(04).
        02  EMPLOYEENUMBER       PIC X(06).
        02  DEPTNUMBER           PIC X(03).
        02  DEPTNAME.
              49  DEPTNAMEL      PIC S9(4)  COMP.
              49  DEPTNAMEC      PIC X(36)  VALUE SPACES.
*
 77  PERCENT-COUNTER             PIC S9(4)  COMP.
```

```
*****************************************************
* SQL DECLARATION FOR VIEW VEMPLP                   *
*****************************************************
           EXEC SQL DECLARE VEMPLP TABLE
                   (EMPLOYEENUMBER   CHAR(06)  NOT NULL,
                    PHONENUMBER      CHAR(04)           )
               END-EXEC.

*****************************************************
* SQL CURSORS                                       *
*****************************************************
*** CURSOR LISTS ALL EMPLOYEE NAMES

          EXEC SQL DECLARE TELE1 CURSOR FOR
                   SELECT *
                   FROM   VPHONE
               END-EXEC.

*** CURSOR LISTS ALL EMPLOYEE NAMES WITH A PATTERN (%) OR (_)
*** FOR LAST NAME

          EXEC SQL DECLARE TELE2 CURSOR FOR
                   SELECT *
                   FROM   VPHONE
                   WHERE  LASTNAME  LIKE :LNAME-WORK
                     AND  FIRSTNAME LIKE :FNAME-WORK
               END-EXEC.

*** CURSOR LISTS ALL EMPLOYEES WITH A SPECIFIC
*** LAST NAME

          EXEC SQL DECLARE TELE3 CURSOR FOR
                   SELECT *
                   FROM   VPHONE
                   WHERE  LASTNAME   =   :LNAME
                     AND  FIRSTNAME LIKE :FNAME-WORK
                END-EXEC.
/
/****************************************************
* FIELDS SENT TO MESSAGE ROUTINE                    *
*****************************************************
 01  MAJOR                     PIC X(07) VALUE 'DSN8BC3'.

 01  MSGCODE                   PIC X(4).

 01  OUTMSG                    PIC X(69).

 01  MSG-REC1.
         02 OUTMSG1            PIC X(69).
         02 RETCODE            PIC S9(9).

 01  MSG-REC2.
         02 OUTMSG2            PIC X(69).


 PROCEDURE DIVISION.
*------------------

*****************************************************
* SQL RETURN CODE HANDLING                          *
*****************************************************
          EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
          EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
          EXEC SQL WHENEVER NOT FOUND  CONTINUE     END-EXEC.

*****************************************************
* MAIN PROGRAM ROUTINE                              *
*****************************************************
 PROG-START.
*                                 **OPEN FILES
          OPEN INPUT  CARDIN
               OUTPUT REPOUT.

*                                 **GET FIRST INPUT
          READ CARDIN RECORD INTO IOAREA
             AT END MOVE 'N' TO INPUT-SWITCH.

*                                 **MAIN ROUTINE
          PERFORM PROCESS-INPUT
             UNTIL NOMORE-INPUT.
 PROG-END.
```

```
*                                  **CLOSE FILES
          CLOSE CARDIN
                REPOUT.
          GOBACK.

*****************************************************
* CREATE REPORT HEADING                             *
* SELECT ACTION                                     *
*****************************************************
 PROCESS-INPUT.
*                                              **PRINT HEADING
          WRITE REPREC FROM REPHDR1
             AFTER ADVANCING TO-TOP-OF-PAGE.
          WRITE REPREC FROM REPHDR2
             AFTER ADVANCING 2 LINES.
          WRITE REPREC FROM REPHDR3.

*                                              **SELECT ACTION
          IF ACTION = 'L'
             PERFORM LIST-FUNCTION
          ELSE
             IF ACTION = 'U'
                PERFORM TELEPHONE-UPDATE

             ELSE
*                                            **INVALID REQUEST
*                                            **PRINT ERROR MESSAGE
          MOVE '068E' TO MSGCODE
          CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
          MOVE OUTMSG TO OUTMSG2
          WRITE REPREC FROM MSG-REC2
                   AFTER ADVANCING 2 LINES.
          READ CARDIN RECORD INTO IOAREA
             AT END MOVE 'N' TO INPUT-SWITCH.
/
*****************************************************
* DETERMINE FORM OF NAME USED TO LIST EMPLOYEES     *
*****************************************************
 LIST-FUNCTION.
*                                            **NO LAST NAME GIVEN
          IF LNAME = SPACES
             MOVE '%' TO LNAME.
*                                            **NO FIRST NAME GIVEN
          IF FNAME = SPACES
             MOVE '%' TO FNAME.
*                                            **LIST ALL EMPLOYEES
          IF LNAME = '*'
             PERFORM LIST-ALL
          ELSE
*                                            **UNSTRING LAST NAME
             UNSTRING LNAME
                DELIMITED BY SPACE
                INTO     LNAME-WORKC
                COUNT IN LNAME-WORKL
*                                            **UNSTRING FIRST NAME
             UNSTRING FNAME
                DELIMITED BY SPACE
                INTO     FNAME-WORKC
                COUNT IN FNAME-WORKL
*                                             **COUNT %'S
             MOVE ZERO TO PERCENT-COUNTER
             INSPECT LNAME
                TALLYING PERCENT-COUNTER FOR ALL '%'
             IF PERCENT-COUNTER > ZERO
*                                          **IF NO %'S THEN
*                                          **LIST SPECIFIC NAME(S)
*                                          **ELSE
*                                          **LIST GENERIC NAME(S)
                PERFORM LIST-GENERIC
             ELSE
                PERFORM LIST-SPECIFIC.
/
*****************************************************
* LIST ALL EMPLOYEES                                *
*****************************************************
 LIST-ALL.
*                                            **OPEN CURSOR
          EXEC SQL OPEN TELE1 END-EXEC.

*                                            **GET EMPLOYEES
          EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
```

```
           IF SQLCODE = NOT-FOUND
      *                                      **NO EMPLOYEE FOUND
      *                                      **PRINT ERROR MESSAGE
             MOVE '008I' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO OUTMSG2
             WRITE REPREC FROM MSG-REC2
                 AFTER ADVANCING 2 LINES
           ELSE
      *                                      **LIST ALL EMPLOYEES
              PERFORM PRINT-AND-GET1
                 UNTIL SQLCODE IS NOT EQUAL TO ZERO.

      *                                      **CLOSE CURSOR
           EXEC SQL CLOSE TELE1 END-EXEC.

       PRINT-AND-GET1.
           PERFORM PRINT-A-LINE.
           EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
      /
      *****************************************************
      * LIST GENERIC EMPLOYEES                            *
      *****************************************************
       LIST-GENERIC.
      *                                      **OPEN CURSOR
           EXEC SQL OPEN  TELE2 END-EXEC.

      *                                      **GET EMPLOYEES
           EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.

           IF SQLCODE = NOT-FOUND
      *                                      **NO EMPLOYEE FOUND
      *                                      **PRINT ERROR MESSAGE
             MOVE '008I' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO OUTMSG2
             WRITE REPREC FROM MSG-REC2
                 AFTER ADVANCING 2 LINES
           ELSE
      *                                 **LIST GENERIC EMPLOYEE(S)
              PERFORM PRINT-AND-GET2
                 UNTIL SQLCODE IS NOT EQUAL TO ZERO.

      *                                      **CLOSE CURSOR
           EXEC SQL CLOSE TELE2 END-EXEC.

       PRINT-AND-GET2.
           PERFORM PRINT-A-LINE.
           EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.
      /
      *****************************************************
      * LIST SPECIFIC EMPLOYEES                           *
      *****************************************************
       LIST-SPECIFIC.
      *                                      **OPEN CURSOR
           EXEC SQL OPEN  TELE3 END-EXEC.

      *                                      **GET EMPLOYEES
           EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.

           IF SQLCODE = NOT-FOUND
      *                                      **NO EMPLOYEE FOUND
      *                                      **PRINT ERROR MESSAGE
             MOVE '008I' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO OUTMSG2
             WRITE REPREC FROM MSG-REC2
                 AFTER ADVANCING 2 LINES
           ELSE
      *                                **LIST SPECIFIC EMPLOYEE(S)
              PERFORM PRINT-AND-GET3
                 UNTIL SQLCODE IS NOT EQUAL TO ZERO.

      *                                      **CLOSE CURSOR
           EXEC SQL CLOSE TELE3 END-EXEC.

       PRINT-AND-GET3.
           PERFORM PRINT-A-LINE.
           EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.
      /
      *****************************************************
      * PRINT A LINE OF INFORMATION FROM DIRECTORY        *
```

```
*****************************************************
 PRINT-A-LINE.
*                                           **GET INFORMATION
          MOVE LASTNAMEC TO RLNAME.
          MOVE FIRSTNAMEC TO RFNAME.
          MOVE MIDDLEINITIAL TO RMIDINIT.
          MOVE PHONENUMBER OF PPHONE TO RPHONE.
          MOVE EMPLOYEENUMBER OF PPHONE TO REMPNO.
          MOVE DEPTNUMBER TO RDEPTNO.
          MOVE DEPTNAMEC TO RDEPTNAME.
*                                           **PRINT INFORMATION
          WRITE REPREC FROM REPDATA
             AFTER ADVANCING 2 LINES.

          MOVE SPACES TO LASTNAMEC
                         FIRSTNAMEC
                         DEPTNAMEC.
/
*****************************************************
* UPDATES PHONE NUMBERS FOR EMPLOYEES               *
*****************************************************
 TELEPHONE-UPDATE.
          EXEC SQL UPDATE VEMPLP
                      SET   PHONENUMBER    = :NEWNO
                      WHERE EMPLOYEENUMBER = :ENO END-EXEC.
          IF SQLCODE = ZERO
*                                         **EMPLOYEE FOUND
*                                         **UPDATE SUCCESSFUL
*                                         **PRINT CONFIRMATION
*                                         **MESSAGE
             MOVE '004I' TO MSGCODE
          ELSE
*                                           **NO EMPLOYEE FOUND
*                                           **UPDATE FAILED
*                                           **PRINT ERROR MESSAGE
             MOVE '007E' TO MSGCODE.
          CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
          MOVE OUTMSG TO OUTMSG2.
          WRITE REPREC FROM MSG-REC2
                AFTER ADVANCING 2 LINES.
/
*****************************************************
* SQL ERROR OCCURRED - GET ERROR MESSAGE            *
*****************************************************
 DBERROR.
*                                           **SQL ERROR
*                                           **PRINT ERROR MESSAGE
          MOVE '060E' TO MSGCODE
          CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
          MOVE OUTMSG TO OUTMSG1 OF  MSG-REC1.
          MOVE SQLCODE TO RETCODE OF MSG-REC1.
          WRITE REPREC FROM MSG-REC1
             AFTER ADVANCING 2 LINES.
          CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
          IF RETURN-CODE = ZERO
             PERFORM ERROR-PRINT VARYING ERROR-INDEX
                FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10
          ELSE

*                                           **MESSAGE FORMAT
*                                           **ROUTINE ERROR
*                                           **PRINT ERROR MESSAGE
             MOVE '075E' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO OUTMSG1 OF MSG-REC1
             MOVE RETURN-CODE TO RETCODE OF MSG-REC1
             WRITE REPREC FROM MSG-REC1
                AFTER ADVANCING 2 LINES.

***********************************************************
* SQL RETURN CODE HANDLING WHEN PROCESSING CANNOT PROCEED *
***********************************************************
          EXEC SQL WHENEVER SQLERROR   CONTINUE     END-EXEC.
          EXEC SQL WHENEVER SQLWARNING CONTINUE     END-EXEC.
          EXEC SQL WHENEVER NOT FOUND  CONTINUE     END-EXEC.

*                                            **PERFORM ROLLBACK
          EXEC SQL ROLLBACK END-EXEC.

          IF SQLCODE = ZERO

*                                           **ROLLBACK SUCCESSFUL
```

```
      *                                           **PRINT CONFIRMATION
      *                                           **MESSAGE
                   MOVE '053I' TO MSGCODE
                ELSE

      *                                            **ROLLBACK FAILED
      *                                           **PRINT ERROR MESSAGE
                   MOVE '061E' TO MSGCODE.
                CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
                MOVE OUTMSG TO OUTMSG1 OF MSG-REC1.
                MOVE SQLCODE TO RETCODE OF MSG-REC1.
                WRITE REPREC FROM MSG-REC1
                      AFTER ADVANCING 2 LINES.
                GO TO PROG-END.

      *****************************************************
      *  PRINT MESSAGE TEXT                               *
      *****************************************************
       ERROR-PRINT.
                WRITE REPREC FROM ERROR-TEXT (ERROR-INDEX)
                   AFTER ADVANCING 1 LINE.
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8BD3

This module lists employee phone numbers and optionally updates them.

```
/*********************************************************************/
/*                                                                   */
/*  Module name = DSN8BD3                                            */
/*                                                                   */
/*  Descriptive name = DB2  SAMPLE APPLICATION                       */
/*                     PHONE APPLICATION                             */
/*                     BATCH                                         */
/*                     C LANGUAGE                                    */
/*                                                                   */
/*     LICENSED MATERIALS - PROPERTY OF IBM                          */
/*     5695-DB2                                                      */
/*     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.      */
/*                                                                   */
/*     STATUS = VERSION 4                                            */
/*                                                                   */
/*  Function = This module lists employee phone numbers and          */
/*             optionally updates them.                              */
/*                                                                   */
/*  Notes = none                                                     */
/*                                                                   */
/*                                                                   */
/*  Module type    = C program                                       */
/*     Processor   = DB2  precompiler, C compiler                    */
/*     Module size = see link edit                                   */
/*     Attributes  = not reentrant or reusable                       */
/*                                                                   */
/*  Entry point = DSN8BD3                                            */
/*     Purpose  = see function                                       */
/*     Linkage  = invoked from DSN command processor subcommand RUN  */
/*     Input    =                                                    */
/*                                                                   */
/*                symbolic label/name = CARDIN                       */
/*                description = INPUT REQUEST FILE                   */
/*                                                                   */
/*                symbolic label/name = VPHONE                       */
/*                description = VIEW OF TELEPHONE TABLE: PHONE       */
/*                                                                   */
/*     Output   =                                                    */
/*                                                                   */
/*                symbolic label/name = REPORT                       */
/*                description = PRINTED REPORT AND RESULTS           */
/*                                                                   */
/*                symbolic label/name = VEMPLP                       */
/*                description = VIEW OF EMPLOYEE INFORMATION         */
/*                                                                   */
/*                                                                   */
/*  Exit-normal = return code 0 normal completion                    */
/*                                                                   */
```

```
/*  Exit-error =                                                     */
/*                                                                   */
/*     Return code    = none                                         */
/*                                                                   */
/*     Abend codes    =  none                                        */
/*                                                                   */
/*     Error-messages =                                              */
/*           DSN8000I - REQUEST IS:  ...                             */
/*           DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED                */
/*           DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE     */
/*           DSN8008I - NO EMPLOYEE FOUND IN TABLE                   */
/*           DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED     */
/*           DSN8060E - SQL ERROR, RETURN CODE IS:                   */
/*           DSN8061E - ROLLBACK FAILED, RETURN CODE IS:             */
/*           DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'        */
/*           DSN8075E - MESSAGE FORMAT ROUTINE ERROR,                */
/*                      RETURN CODE IS:                              */
/*                                                                   */
/*  External references  =                                           */
/*     Routines/services =                                           */
/*           DSNTIAR  - translate sqlca into messages                */
/*                                                                   */
/*     Data-areas        = none                                      */
/*                                                                   */
/*     Control-blocks    =                                           */
/*           SQLCA    - sql communication area                       */
/*                                                                   */
/*  Tables               = none                                      */
/*                                                                   */
/*  Change-activity      =                                           */
/*  10/03/94 Updated cardin statement to prevent looping. KEW1351  @51*/
/*                                                        PN61293  @51*/
/*                                                                   */
/* *Pseudocode*                                                      */
/*                                                                   */
/* main:                                                             */
/*   do while more input                                             */
/*     get input                                                     */
/*     display request                                               */
/*     process request                                               */
/*   end                                                             */
/*                                                                   */
/* Do_req:                                                           */
/*   case (action)                                                   */
/*                                                                   */
/*     subcase ('L')                                                 */
/*       create report heading                                       */
/*       if lastname is '*' then                                     */
/*         list all employees                                        */
/*       else                                                        */
/*         if lastname contains '%' then                             */
/*         list employees generic                                    */
/*         else                                                      */
/*           list employees specific                                 */
/*     endsub                                                        */
/*                                                                   */
/*     subcase ('U')                                                 */
/*       update phonenumber for employee                             */
/*       write confirmation message                                  */
/*                                                                   */
/*     otherwise                                                     */
/*       invalid request                                             */
/*     endsub                                                        */
/*                                                                   */
/*   endcase                                                         */
/*                                                                   */
/* Prt_row:                                                          */
/*  print a row of the report                                        */
/*                                                                   */
/* Sql_err:                                                          */
/*   if sql error occurs then                                        */
/*     rollback                                                      */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/* Include C library definitions                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
/**********************************************************************/
/* General declarations                                               */
/**********************************************************************/
#define NOTFOUND 100

/**********************************************************************/
/* Input / Output files                                               */
/**********************************************************************/
FILE *cardin;                       /* Input control cards         */
FILE *report;                       /* Output phone report         */

/**********************************************************************/
/* Input record structure                                             */
/**********************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
struct {
  char action[2];                   /* L for list or U for update  */
  char lname[16];                   /* last name or pattern- L mode*/
  char fname[13];                   /* first name or pattern-L mode*/
  char eno[7];                      /* employee number- U mode     */
  char newno[5];                    /* new phone number- U mode    */
       } ioarea;

char trail[43];                     /* unused portion of input rec */
char slname[16];                    /* unmodified last name pattern*/
EXEC SQL END DECLARE SECTION;

/**********************************************************************/
/* Report headings                                                    */
/**********************************************************************/
struct {
  char hdr011[30];
  char hdr012[32];
       } hdr0 = {
                      "    REQUEST  LAST NAME        ",
                      "FIRST NAME    EMPNO   NEW XT.NO"};
#define rpthdr0 hdr0.hdr011

struct {
  char hdr111[29];
  char hdr112[21];
  char hdr113[30];
       } hdr1 = {
                      "----------------------------",
                      " TELEPHONE DIRECTORY ",
                      "-----------------------------"};
#define rpthdr1 hdr1.hdr111

struct {
  char hdr211[10];
  char hdr212[11];
  char hdr213[ 8];
  char hdr214[ 6];
  char hdr215[ 9];
  char hdr216[ 5];
  char hdr217[ 5];
  char hdr221[ 7];
  char hdr222[ 7];
  char hdr223[ 5];
  char hdr224[ 5];
  char hdr225[ 5];
       } hdr2 = {
                      "LAST NAME",
                      "FIRST NAME",
                      "INITIAL",
                      "PHONE",
                      "EMPLOYEE",
                      "WORK",
                      "WORK",
                      "NUMBER",
                      "NUMBER",
                      "DEPT",
                      "DEPT",
                      "NAME"
                };
#define rpthdr2 hdr2.hdr211,hdr2.hdr212,hdr2.hdr213,hdr2.hdr214,\
                hdr2.hdr215,hdr2.hdr216,hdr2.hdr217,hdr2.hdr221,\
                hdr2.hdr222,hdr2.hdr223,hdr2.hdr224,hdr2.hdr225

/**********************************************************************/
/* Report formats                                                     */
/**********************************************************************/
```

```
static char fmt1[] = "\n %s\n";
static char fmt2[] = " %9s%17s%10s%6s%10s%5s%5s\n%43s%7s%7s%5s%5s\n";
static char fmt3[] = " %-16s%-16s%-5s%-7s%-9s%-5s%-36s\n";
static char fmt4[] = " %1c%15c%12c%6c%4c%43c";
static char fmt5[] =
   "\n\n %s\n %s\n   --%-7s--%-15s--%-12s--%-5s--%-9s--\n";

/**********************************************************************/
/* Fields sent to message routine                                     */
/**********************************************************************/
char outmsg[70];                       /* error/information msg buffer*/
char module[ 8] = "DSN8BD3";           /* module name for message rtn */

extern DSN8MDG();                      /* message routine             */

/**********************************************************************/
/* SQL communication area                                             */
/**********************************************************************/
EXEC SQL INCLUDE SQLCA;

/**********************************************************************/
/* SQL declaration for view VPHONE                                    */
/**********************************************************************/
EXEC SQL DECLARE VPHONE TABLE
       (LASTNAME       VARCHAR(15)  NOT NULL,
        FIRSTNAME      VARCHAR(12)  NOT NULL,
        MIDDLEINITIAL     CHAR( 1)  NOT NULL,
        PHONENUMBER       CHAR( 4)          ,
        EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
        DEPTNUMBER        CHAR( 3)  NOT NULL,
        DEPTNAME       VARCHAR(36)  NOT NULL);

/**********************************************************************/
/* Structure for pphone record                                        */
/**********************************************************************/

/* Note: since the sample program data does not contain imbedded      */
/*       nulls, the C language null terminated string can be used to  */
/*       receive the varchar fields from DB2.                         */

EXEC SQL BEGIN DECLARE SECTION;
struct {
  char lastname[16];
  char firstname[13];
  char middleinitial[2];
  char phonenumber[5];
  char employeenumber[7];
  char deptnumber[4];
  char deptname[37];
  } pphone;
EXEC SQL END   DECLARE SECTION;

/**********************************************************************/
/* SQL declaration for view VEMPLP (used for update processing)       */
/**********************************************************************/
EXEC SQL DECLARE VEMPLP TABLE
       (EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
        PHONENUMBER       CHAR( 4)          );

/**********************************************************************/
/* Structure for pemplp record                                        */
/**********************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
struct {
  char employeenumber[7];
  char phonenumber[5];
       } pemplp;
EXEC SQL END   DECLARE SECTION;

/**********************************************************************/
/* SQL cursors                                                        */
/**********************************************************************/
/* cursor to list all employee names */
EXEC SQL DECLARE TELE1 CURSOR FOR
         SELECT *
         FROM VPHONE;

/* cursor to list all employee names with a pattern */
/* (%) or (_) in last name                          */
EXEC SQL DECLARE TELE2 CURSOR FOR
         SELECT *
         FROM VPHONE
```

```
         WHERE LASTNAME LIKE :lname;

/* cursor to list all employees with a specific last name */
EXEC SQL DECLARE TELE3 CURSOR FOR
         SELECT *
         FROM VPHONE
         WHERE LASTNAME = :slname
           AND FIRSTNAME LIKE :fname;

/**********************************************************************/
/* SQL return code handling                                           */
/**********************************************************************/
EXEC SQL WHENEVER SQLERROR   GOTO DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/**********************************************************************/
/* main program routine                                               */
/**********************************************************************/
extern main()
{
  /* Open the input and output files */
  cardin = fopen("DD:CARDIN","r,recfm=FB,lrecl=80,blksize=80"); /*@51*/
  report = fopen("DD:REPORT","w");

  /* While more input, process */
  while (!feof(cardin))
  {
    /* Read the next request */
    if (fscanf(cardin, fmt4,
                        ioarea.action,
                        ioarea.lname,
                        ioarea.fname,
                        ioarea.eno,
                        ioarea.newno,
                        trail) == 6)
    {
      /* Display the request */
      DSN8MDG(module, "000I", outmsg);
      fprintf(report, fmt5,
                        outmsg,
                        rpthdr0,
                        ioarea.action,
                        ioarea.lname,
                        ioarea.fname,
                        ioarea.eno,
                        ioarea.newno);
      Do_req();
    }
  } /* endwhile */
  fclose(report);
} /* end main */

/**********************************************************************/
/* Process the current request                                        */
/**********************************************************************/
Do_req()
{
  char *blankloc;                    /* string translation pointer  */
  strcpy(slname, ioarea.lname);      /* save untranslated last name */
  while (blankloc = strpbrk(ioarea.lname, " "))
    *blankloc = '%';                 /* translate blanks into %     */
  while (blankloc = strpbrk(ioarea.fname, " "))
    *blankloc = '%';                 /* translate blanks into %     */

  /* Determine request type */
  switch (ioarea.action[0])
  {

    /* Process LIST request */
    case 'L':
      /* Print the report headings */
      fprintf(report, fmt1, rpthdr1);
      fprintf(report, fmt2, rpthdr2);

      /* List all employees */
      if (!strcmp(slname,"*              ")){
        EXEC SQL OPEN TELE1;
        EXEC SQL FETCH TELE1 INTO :pphone;
        if (sqlca.sqlcode == NOTFOUND){          /* If no employees   */
          DSN8MDG(module, "008I", outmsg);       /* found, display    */
          fprintf(report, " %s\n", outmsg);      /* error message     */
```

```
        } /* endif */
        while (sqlca.sqlcode == 0){
          Prt_row();
          EXEC SQL FETCH TELE1 INTO :pphone;
        } /* endwhile */
        EXEC SQL CLOSE TELE1;

        /* List generic employees */
      } else {
        if (strpbrk(slname, "%")) {
          EXEC SQL OPEN TELE2;
          EXEC SQL FETCH TELE2 INTO :pphone;
          if (sqlca.sqlcode == NOTFOUND){        /* If no employees   */
            DSN8MDG(module, "008I", outmsg);     /* found, display    */
            fprintf(report, " %s\n", outmsg);    /* error message     */
          } else {
            while (sqlca.sqlcode == 0){
              Prt_row();
              EXEC SQL FETCH TELE2 INTO :pphone;
            } /* endwhile */
          } /* endif */
          EXEC SQL CLOSE TELE2;

          /* List specific employee */
        } else {
          EXEC SQL OPEN TELE3;
          EXEC SQL FETCH TELE3 INTO :pphone;
          if (sqlca.sqlcode == NOTFOUND){        /* If no employee    */
            DSN8MDG(module, "008I", outmsg);     /* found, display    */
            fprintf(report, " %s\n", outmsg);    /* error message     */
          } else {
            while (sqlca.sqlcode == 0){
              Prt_row();
              EXEC SQL FETCH TELE3 INTO :pphone;
            } /* endwhile */
          } /* endif */
          EXEC SQL CLOSE TELE3;
        } /* endif */
      } /* endif */
      break; /* end of 'L' request */

    /* Update an employee phone number */
    case 'U':
      EXEC SQL UPDATE VEMPLP
               SET PHONENUMBER = :ioarea.newno
               WHERE EMPLOYEENUMBER = :ioarea.eno;
      if (sqlca.sqlcode == 0){                   /* If employee       */
        DSN8MDG(module, "004I", outmsg);         /* updated, display  */
        fprintf(report, " %s\n", outmsg);        /* confirmation msg  */
      } else {
        DSN8MDG(module, "007E", outmsg);         /* otherwise, display*/
        fprintf(report, " %s\n", outmsg);        /* error message     */
      } /* endif */
      break;

    /* Invalid request type */
    default:
      DSN8MDG(module, "068E", outmsg);           /* Display error msg */
      fprintf(report, " %s\n", outmsg);
  } /* endswitch */
  return;

DBERROR:
  Sql_err();
} /* end Do_req */

/**********************************************************************/
/* Print a single employee on the report                              */
/**********************************************************************/
Prt_row()
{
  fprintf(report, fmt3, pphone.lastname,
                        pphone.firstname,
                        pphone.middleinitial,
                        pphone.phonenumber,
                        pphone.employeenumber,
                        pphone.deptnumber,
                        pphone.deptname);
}

/**********************************************************************/
/* SQL error handler                                                  */
```

```
/*******************************************************************/
#pragma linkage(dsntiar, OS)
Sql_err() {
#define data_len 120
#define data_dim 10
struct error_struct {
  short int error_len;
  char error_text[data_dim][data_len];
  } error_message = {data_dim * data_len};
extern short int dsntiar(struct sqlca        *sqlca,
                         struct error_struct *msg,
                         int                 *len);

short int rc;
int i;
static int lrecl = data_len;

  DSN8MDG(module, "060E", outmsg);
  fprintf(report, " %s %i\n", outmsg, sqlca.sqlcode);
  rc = dsntiar(&sqlca, &error_message, &lrecl);  /* Format the sqlca  */
  if (rc == 0){                                  /* Print formatted   */
    for (i=0;i<=7;i++){                          /*   sqlca           */
      fprintf(report, "%.120s\n", error_message.error_text  [i]);
    } /* endfor */
  } else {
    DSN8MDG(module, "075E", outmsg);
    fprintf(report, " %s %hi\n", outmsg, rc);
  } /* endif */

  /* Attempt to rollback any work already done */
  EXEC SQL WHENEVER SQLERROR   CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL WHENEVER NOT FOUND  CONTINUE;

  EXEC SQL ROLLBACK;
  if (sqlca.sqlcode == 0){                       /* If rollback       */
    DSN8MDG(module, "053I", outmsg);             /* completed, display*/
    fprintf(report, " %s\n", outmsg);            /* confirmation msg  */
  } else {                                       /* otherwise, display*/
    DSN8MDG(module, "061E", outmsg);             /* error message     */
    fprintf(report, " %s %i\n", outmsg, sqlca.sqlcode);
  } /* endif */
  fclose(report);
  exit(0);
} /* end of Sql_err */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8BE3

This module uses the class emp_db2 to list or update employee phone numbers from a Db2 database .

```
/*******************************************************************/
/*                                                                 */
/*  Module name = DSN8BD3                                          */
/*                                                                 */
/*  Descriptive name = DB2  SAMPLE APPLICATION                     */
/*                       PHONE APPLICATION                         */
/*                       BATCH                                     */
/*                       C++ LANGUAGE                              */
/*                                                                 */
/*    LICENSED MATERIALS - PROPERTY OF IBM                         */
/*    5625-DB2                                                     */
/*    (C) COPYRIGHT 1982, 2003 IBM CORP.  ALL RIGHTS RESERVED.     */
/*                                                                 */
/*    STATUS = VERSION 8                                           */
/*                                                                 */
/*  Function = This module uses the class emp_db2 to list or update */
/*             employee phone numbers from a DB2 database          */
/*                                                                 */
/*  Module type    = C++ program                                   */
/*     Processor   = DB2  precompiler, C++ compiler                */
/*     Module size = see link edit                                 */
/*     Attributes  = not reentrant or reusable                     */
/*                                                                 */
/*  Entry point = DSN8BD3                                          */
```

```
/*     Purpose  = see function                                        */
/*     Linkage  = invoked from DSN command processor subcommand RUN  */
/*                                                                    */
/*     Input    = symbolic label/name = CARDIN                       */
/*                description = INPUT REQUEST FILE                   */
/*                                                                    */
/*     Output   = symbolic label/name = REPORT                       */
/*                description = PRINTED REPORT AND RESULTS           */
/*                                                                    */
/*  Exit-normal = return code 0 normal completion                    */
/*                                                                    */
/*  Exit-error =                                                      */
/*                                                                    */
/*     Return code    = none                                          */
/*                                                                    */
/*     Abend codes    = none                                          */
/*                                                                    */
/*     Error-messages =                                               */
/*           DSN8000I - REQUEST IS:  ...                              */
/*           DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'        */
/*                      RETURN CODE IS:                               */
/*                                                                    */
/*  External references  =                                            */
/*     Routines/services = none                                       */
/*                                                                    */
/*     Data-areas        = none                                       */
/*                                                                    */
/*     Control-blocks    = none                                       */
/*                                                                    */
/*  Tables               = none                                       */
/*                                                                    */
/*  Change-activity      =                                            */
/*      02/05/96 Katja      KFD0024  C++ sample (D9031)              */
/*                                   Created based on C sample       */
/*                                                                    */
/*********************************************************************/

#include <string.h>
/*********************************************************************/
/* Include emp_db2 C++ class definition                              */
/*    (includes other global declarations)                           */
/*********************************************************************/
#include "DSN8BEH"

  /*******************************************************************/
  /* Input record structure                                          */
  /*******************************************************************/
  struct {
    char action[2];                /* L for list or U for update     */
    char lname[16];                /* last name or pattern- L mode   */
    char fname[13];                /* first name or pattern-L mode   */
    char eno[7];                   /* employee number- U mode        */
    char newno[5];                 /* new phone number- U mode       */
  } ioarea;

  char slname[16];                 /* unmodified last name pattern   */
  class emp_db2 proc1;             /* DB2 employee object           */

  /*******************************************************************/
  /* Function to process the current request                         */
  /*******************************************************************/
  void Do_req(FILE *outfile)
  {
    char *blankloc;                /* string translation pointer     */

    /*****************************************************************/
    /* Report headings                                               */
    /*****************************************************************/
    struct
    {
      char hdr111[30];
      char hdr112[22];
      char hdr113[30];
    } hdr1 = {
             "----------------------------",
             " TELEPHONE DIRECTORY ",
             "----------------------------"};
    #define rpthdr1 hdr1.hdr111,hdr1.hdr112,hdr1.hdr113

    struct
    {
      char hdr211[10];
```

```
      char hdr212[11];
      char hdr213[ 8];
      char hdr214[ 6];
      char hdr215[ 9];
      char hdr216[ 5];
      char hdr217[ 5];
      char hdr221[ 7];
      char hdr222[ 7];
      char hdr223[ 5];
      char hdr224[ 5];
      char hdr225[ 5];
    } hdr2 = {
               "LAST NAME",
               "FIRST NAME",
               "INITIAL",
               "PHONE",
               "EMPLOYEE",
               "WORK",
               "WORK",
               "NUMBER",
               "NUMBER",
               "DEPT",
               "DEPT",
               "NAME"};
    #define rpthdr2 hdr2.hdr211,hdr2.hdr212,hdr2.hdr213,hdr2.hdr214,\
                    hdr2.hdr215,hdr2.hdr216,hdr2.hdr217,hdr2.hdr221,\
                    hdr2.hdr222,hdr2.hdr223,hdr2.hdr224,hdr2.hdr225

    /****************************************************************/
    /* Report formats                                               */
    /****************************************************************/
    static char fmt1[] = "\n %s\n %s\n %s\n";
    static char fmt2[] =
     " %9s%17s%10s%6s%10s%5s%5s\n%43s%8s%7s%5s%5s\n";

    /****************************************************************/
    /* Start processing input record                                */
    /****************************************************************/
    strcpy(slname, ioarea.lname);  /* save untranslated last name    */
    while (blankloc = strpbrk(ioarea.lname, " "))
      *blankloc = '%';             /* translate blanks into %        */
    while (blankloc = strpbrk(ioarea.fname, " "))
      *blankloc = '%';             /* translate blanks into %        */

    /* Determine request type */
    switch (ioarea.action[0])
    {

      /* Process LIST request */
      case 'L':
        /* Print the report headings */
        fprintf(outfile, fmt1, rpthdr1);
        fprintf(outfile, fmt2, rpthdr2);

        if (!strcmp(slname,"*              "))
          /* List all employees */
          proc1.Listall(outfile);
        else
        {
          if (strpbrk(slname, "%"))
            /* List generic employees */
            proc1.Listsome(outfile,ioarea.lname);
          else
            /* List specific employee */
            proc1.Listone(outfile,slname,ioarea.fname);
        } /* else - list selected employees */
        break; /* end 'L' request */

      /* Update an employee phone number */
      case 'U':
        proc1.Empupdate(outfile,ioarea.newno,ioarea.eno);
        break;

      /* Invalid request type */
      default:
        DSN8MDG(module, "068E", outmsg);        /* Display error msg */
        fprintf(outfile, " %s\n", outmsg);
    } /* endswitch */
    return;
  } /* end Do_req */

  /******************************************************************/
```

```
  /* Function to read a request from an open file                    */
  /*******************************************************************/
  int Read_req(FILE *infile)
  {
    static char fmt4[] = " %1c%15c%12c%6c%4c%43c";   /* input format */
    char trail[43];                /* unused part of input record    */
    char *newlloc;                 /* addr of newline char in field  */

    strcpy(ioarea.action," ");
    strcpy(ioarea.lname,"               ");
    strcpy(ioarea.fname,"            ");
    strcpy(ioarea.eno,"      ");
    strcpy(ioarea.newno,"      ");
    /* Read the next request */
    if (fscanf(infile, fmt4,
                        ioarea.action,
                        ioarea.lname,
                        ioarea.fname,
                        ioarea.eno,
                        ioarea.newno,
                        trail) == 6)
    {
      if ((newlloc = strpbrk(ioarea.lname, "\n")) != NULL)
        *newlloc = ' ';            /* change to blank for now        */
      if ((newlloc = strpbrk(ioarea.fname, "\n")) != NULL)
        *newlloc = ' ';            /* change to blank for now        */
      ioarea.eno[6] = '\0';
      ioarea.newno[4] = '\0';
      return 0;
    }
    else
      return 1;
  } /* end Read_req */

  /*******************************************************************/
  /* Function to echo a request                                      */
  /*******************************************************************/
  void Echo_req(FILE *outfile)
  {
    /* Local declarations */
    struct                                      /* report header     */
    {
      char hdr011[31];
      char hdr012[33];
    } hdr0 = {
              "    REQUEST  LAST NAME         ",
              "FIRST NAME    EMPNO    NEW XT.NO"};
    #define rpthdr0 hdr0.hdr011

    static char fmt5[] =                        /* output format     */
     "\n\n %s\n %s%s\n   --%-7s--%-15s--%-12s--%-7s--%-9s--\n";
    /* End local declarations */

    /* Display the request */
    DSN8MDG(module, "000I", outmsg);
    fprintf(outfile, fmt5,
                          outmsg,
                          hdr0.hdr011,
                          hdr0.hdr012,
                          ioarea.action,
                          ioarea.lname,
                          ioarea.fname,
                          ioarea.eno,
                          ioarea.newno);
      return;
  } /* end Echo_req */

/*********************************************************************/
/* main program routine                                              */
/*********************************************************************/
extern main()
{
  int retcode;
  FILE *cardin;                    /* Input control cards            */
  FILE *report;                    /* Output phone report            */

  /* Open the input and output files */
  cardin = fopen
            ("DD:CARDIN","r,recfm=fb,lrecl=80,blksize=80");
  report = fopen("DD:REPORT","w");

  /* While more input, process */
```

```
    while (!feof(cardin))
    {
      /* Read the next request */
      retcode = Read_req(cardin);
      if (retcode == 0)
      {
        /* Display the request */
        Echo_req(report);
        Do_req(report);
      }
    } /* endwhile */
    fclose(report);
  } /* end main */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8BP3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
DSN8BP3: PROC REORDER OPTIONS(MAIN);
/*********************************************************************
*                                                                    *
*   MODULE NAME = DSN8BP3                                            *
*                                                                    *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                       *
*                       PHONE APPLICATION                            *
*                       BATCH                                        *
*                       PL/I                                         *
*                                                                    *
*     LICENSED MATERIALS - PROPERTY OF IBM                           *
*     5695-DB2                                                       *
*     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.       *
*                                                                    *
*     STATUS = VERSION 4                                             *
*                                                                    *
*   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND          *
*              UPDATES THEM IF DESIRED.                              *
*                                                                    *
*   NOTES = NONE                                                     *
*                                                                    *
*                                                                    *
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                            *
*      PROCESSOR = DB2  PRECOMPILER, PL/I OPTIMIZER                  *
*      MODULE SIZE = SEE LINK EDIT                                   *
*      ATTRIBUTES = REENTRANT                                        *
*                                                                    *
*   ENTRY POINT = DSN8BP3                                            *
*      PURPOSE = SEE FUNCTION                                        *
*      LINKAGE = INVOKED FROM DSN RUN                                *
*      INPUT   =                                                     *
*                                                                    *
*                SYMBOLIC LABEL/NAME = CARDIN                        *
*                DESCRIPTION = INPUT REQUEST FILE                    *
*                                                                    *
*                SYMBOLIC LABEL/NAME = VPHONE                        *
*                DESCRIPTION = VIEW OF TELEPHONE INFORMATION         *
*                                                                    *
*      OUTPUT  =                                                     *
*                                                                    *
*                SYMBOLIC LABEL/NAME = REPORT                        *
*                DESCRIPTION = REPORT OF EMPLOYEE PHONE NUMBERS      *
*                                                                    *
*                SYMBOLIC LABEL/NAME = VEMPLP                        *
*                DESCRIPTION = VIEW OF EMPLOYEE INFORMATION          *
*                                                                    *
*                                                                    *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION                    *
*                                                                    *
*   EXIT-ERROR =                                                     *
*                                                                    *
*      RETURN CODE = NONE                                            *
*                                                                    *
*      ABEND CODES =  NONE                                           *
*                                                                    *
*      ERROR-MESSAGES =                                              *
```

```
 *            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED                *
 *            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE     *
 *            DSN8008I - NO EMPLOYEE FOUND IN TABLE                   *
 *            DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED     *
 *            DSN8060E - SQL ERROR, RETURN CODE IS:                   *
 *            DSN8061E - ROLLBACK FAILED, RETURN CODE IS:             *
 *            DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'        *
 *            DSN8075E - MESSAGE FORMAT ROUTINE ERROR,                *
 *                       RETURN CODE IS :                             *
 *                                                                    *
 *   EXTERNAL REFERENCES =                                            *
 *      ROUTINES/SERVICES =                                           *
 *         DSN8MPG            - ERROR MESSAGE ROUTINE                 *
 *                                                                    *
 *      DATA-AREAS = NONE                                             *
 *                                                                    *
 *      CONTROL-BLOCKS =                                              *
 *         SQLCA               - SQL COMMUNICATION AREA               *
 *                                                                    *
 *   TABLES = NONE                                                    *
 *                                                                    *
 *   CHANGE-ACTIVITY = NONE                                           *
 *                                                                    *
 *                                                                    *
 *  *PSEUDOCODE*                                                      *
 *                                                                    *
 *  PROCEDURE                                                         *
 *    GET FIRST INPUT                                                 *
 *    DO WHILE MORE INPUT                                             *
 *      CREATE REPORT HEADING                                         *
 *      CASE (ACTION)                                                 *
 *                                                                    *
 *        SUBCASE ('L')                                               *
 *          IF LASTNAME IS '*' THEN                                   *
 *            LIST ALL EMPLOYEES                                      *
 *          ELSE                                                      *
 *            IF LASTNAME CONTAINS '%' THEN                           *
 *              LIST EMPLOYEES GENERIC                                *
 *            ELSE                                                    *
 *              LIST EMPLOYEES SPECIFIC                               *
 *        ENDSUB                                                      *
 *                                                                    *
 *        SUBCASE ('U')                                               *
 *          UPDATE PHONENUMBER FOR EMPLOYEE                           *
 *          WRITE CONFIRMATION MESSAGE                                *
 *        OTHERWISE                                                   *
 *          INVALID REQUEST                                           *
 *        ENDSUB                                                      *
 *                                                                    *
 *      ENDCASE                                                       *
 *      GET NEXT INPUT                                                *
 *    END                                                             *
 *                                                                    *
 *    IF SQL ERROR OCCURS THEN                                        *
 *      ROLLBACK                                                      *
 *  END.                                                              *
 *                                                                    *
 *-------------------------------------------------------------------*/
1/**********************/
 /* INPUT/OUTPUT FILES */
 /**********************/

 DCL CARDIN FILE STREAM INPUT;         /* INPUT CONTROL CARDS */
 DCL REPORT FILE STREAM OUTPUT PRINT;  /* OUTPUT PHONE REPORT */

 /********************/
 /* ENDFILE HANDLING */
 /********************/

 ON ENDFILE (CARDIN)  EOF = '1'B;

 /***********************/
 /* STRUCTURE FOR INPUT */
 /***********************/

 DCL 1 IOAREA,
      2 ACTION CHAR( 1),                                /* ACTION */
      2 LNAME  CHAR(15),                             /* LAST NAME */
      2 FNAME  CHAR(12),                            /* FIRST NAME */
      2 ENO    CHAR( 6),                       /* EMPLOYEE NUMBER */
      2 NEWNO  CHAR( 4);                          /* PHONE NUMBER */
```

```
 /***********************/
 /* WORK VARIABLES      */
 /***********************/

 DCL   LNAMEWK CHAR(15) VAR;    /* WORK VERSION OF LAST NAME      */
 DCL   FNAMEWK CHAR(12) VAR;    /* WORK VERSION OF FIRST NAME     */

 /***************************/
 /* REPORT HEADER STRUCTURE */
 /***************************/

 DCL 1 REPHDR1 STATIC,
     2 HDR111          CHAR(29) INIT ((29)'-'),
     2 HDR112          CHAR(21) INIT (' TELEPHONE DIRECTORY '),
     2 HDR113          CHAR(28) INIT ((28)'-');
 DCL 1 REPHDR2 STATIC,
     2 HDR211          CHAR( 9) INIT ('LAST NAME'),
     2 HDR212          CHAR(10) INIT ('FIRST NAME'),
     2 HDR213          CHAR( 7) INIT ('INITIAL'),
     2 HDR214          CHAR( 5) INIT ('PHONE'),
     2 HDR215          CHAR( 8) INIT ('EMPLOYEE'),
     2 HDR216          CHAR( 4) INIT ('WORK'),
     2 HDR217          CHAR( 4) INIT ('WORK'),
     2 HDR221          CHAR( 6) INIT ('NUMBER'),
     2 HDR222          CHAR( 6) INIT ('NUMBER'),
     2 HDR223          CHAR( 4) INIT ('DEPT'),
     2 HDR224          CHAR( 4) INIT ('DEPT'),
     2 HDR225          CHAR( 4) INIT ('NAME');

 /******************/
 /* REPORT FORMATS */
 /******************/

 L1: FORMAT (A(29),A(21),A(28));
 L2: FORMAT (SKIP(2),A(9),X(7),A(10),X(3),A(7),X(1),A(5),X(2),A(8),
             X(1),A(4),X(1),A(4),SKIP,X(37),A(6),X(1),A(6),X(3),
             A(4),X(1),A(4),X(1),A(4));
 L3: FORMAT (SKIP,A(15),X(1),A(12),X(4),A(1),X(4),A(4),X(3),A(6),X(3),
             A(3),X(2),A(36));
 L4: FORMAT (COL(1),A(1),A(15),A(12),A(6),A(4));

 /*********************************/
 /* FIELDS SENT TO MESSAGE ROUTINE*/
 /*********************************/

 DCL OUTMSG     CHAR(69);
 DCL MODULE     CHAR(07) INIT('DSN8BP3');

 DCL DSN8MPG EXTERNAL ENTRY;

 /********************/
 /* GENERAL DECLARES */
 /********************/

 DCL (ADDR,
      DIM,
      PLIRETV,
      TRANSLATE,
      INDEX) BUILTIN;
 DCL  EOF BIT(1) INIT ('0'B);
 DCL  I   BIN FIXED(15);
 DCL  ZERO BIN FIXED(15) STATIC INIT(0);
 DCL  ONE  BIN FIXED(15) STATIC INIT(1);
 DCL  NOTFOUND BIN FIXED(15) STATIC INIT(100);

1/***********************************/
 /* SQL DECLARATION FOR VIEW VPHONE */
 /***********************************/

 EXEC SQL DECLARE VPHONE TABLE
       (LASTNAME       VARCHAR(15)  NOT NULL,
        FIRSTNAME      VARCHAR(12)  NOT NULL,
        MIDDLEINITIAL     CHAR( 1)  NOT NULL,
        PHONENUMBER       CHAR( 4)          ,
        EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
        DEPTNUMBER        CHAR( 3)  NOT NULL,
        DEPTNAME       VARCHAR(36)  NOT NULL);

 /**************************/
 /* SQL COMMUNICATION AREA */
 /**************************/
```

```
 EXEC SQL INCLUDE SQLCA;

 /*******************************/
 /* STRUCTURE FOR PPHONE RECORD */
 /*******************************/

 DCL 1 PPHONE,
      2 LASTNAME       CHAR(15) VAR,
      2 FIRSTNAME      CHAR(12) VAR,
      2 MIDDLEINITIAL  CHAR( 1),
      2 PHONENUMBER    CHAR( 4),
      2 EMPLOYEENUMBER CHAR( 6),
      2 DEPTNUMBER     CHAR( 3),
      2 DEPTNAME       CHAR(36) VAR;

 /***********************************/
 /* SQL DECLARATION FOR VIEW VEMPLP */
 /***********************************/

 EXEC SQL DECLARE VEMPLP TABLE
        (EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
         PHONENUMBER       CHAR( 4)          );

 /*******************************/
 /* STRUCTURE FOR PEMPLP RECORD */
 /*******************************/

 DCL 1 PEMPLP,
      2 EMPLOYEENUMBER   CHAR(6),
      2 PHONENUMBER CHAR(4);

 /***************/
 /* SQL CURSORS */
 /***************/

 /* CURSOR LISTS ALL EMPLOYEE NAMES */

 EXEC SQL DECLARE TELE1 CURSOR FOR
          SELECT *
          FROM VPHONE;

 /* CURSOR LISTS ALL EMPLOYEE NAMES WITH A PATTERN (% OR _)   */
 /* IN LAST NAME OR A BLANK LAST NAME.                        */

 EXEC SQL DECLARE TELE2 CURSOR FOR
          SELECT *
          FROM VPHONE
          WHERE LASTNAME LIKE :LNAMEWK
          AND FIRSTNAME LIKE :FNAMEWK;

 /* CURSOR LISTS ALL EMPLOYEES WITH A SPECIFIC LAST NAME */

 EXEC SQL DECLARE TELE3 CURSOR FOR
          SELECT *
          FROM VPHONE
          WHERE LASTNAME = :LNAMEWK
          AND FIRSTNAME LIKE :FNAMEWK;
 /****************************/
 /* SQL RETURN CODE HANDLING */
 /****************************/

 EXEC SQL WHENEVER SQLERROR   GOTO DBERROR;
 EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
 EXEC SQL WHENEVER NOT FOUND  CONTINUE;
1/****************************/
 /* MAIN PROGRAM ROUTINE     */
 /****************************/
   GET FILE (CARDIN) EDIT (IOAREA) (R(L4)); /* READ FIRST REQUEST    */
                                    /* PROCESS INPUT REQUESTS     */
   DO WHILE (^EOF);                 /* CONTINUE WHILE MORE TO DO  */
                                    /* PUT REPORT HEADINGS        */
 /****************************/
 /* CREATE REPORT HEADING    */
 /* SELECT ACTION            */
 /****************************/
     PUT FILE (REPORT) PAGE EDIT (REPHDR1) (R(L1));
     PUT FILE (REPORT) EDIT (REPHDR2) (R(L2));
     IF INDEX(LNAME,' ') > 0 THEN
       LNAMEWK = SUBSTR(LNAME,1,INDEX(LNAME,' ')-1);
     ELSE
       LNAMEWK = LNAME;
     IF INDEX(FNAME,' ') > 0 THEN
```

```
      FNAMEWK = SUBSTR(FNAME,1,INDEX(FNAME,' ')-1);
    ELSE
      FNAMEWK = FNAME;
                                    /* GET WORKING VERSIONS OF    */
                                    /* LAST AND FIRST NAMES WITH  */
                                    /* NO TRAILING BLANKS         */
    IF LNAME = ' ' THEN LNAMEWK='%';  /* BLANK NAMES IN INPUT MEAN  */
    IF FNAME = ' ' THEN FNAMEWK='%';  /* SEARCH FOR ALL NAMES       */

    SELECT (ACTION);                /* DETERMINE INPUT REQUEST    */

/***************************************/
/* LIST ALL EMPLOYEES                  */
/***************************************/
      WHEN ('L') DO;                /* LIST EMPLOYEES             */
        IF LNAME = '*' THEN         /* LIST ALL EMPLOYEES         */
          DO;
            EXEC SQL OPEN TELE1;    /* OPEN CURSOR FOR SEARCH     */
            EXEC SQL FETCH TELE1 INTO :PPHONE;/* GET FIRST RECORD   */

            IF SQLCODE = NOTFOUND THEN    /* NO RECORDS FOUND       */
              DO;                         /* GET ERROR MESSAGE      */
               CALL DSN8MPG (MODULE, '008I', OUTMSG);
               PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
              END;

                                    /* GET AND PRINT ALL RECORDS  */
            DO WHILE (SQLCODE = ZERO);
              PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
              EXEC SQL FETCH TELE1 INTO :PPHONE;/* GET NEXT RECORD  */
            END;                                  /* END DO WHILE */

            EXEC SQL CLOSE TELE1;     /* CLOSE CURSOR FOR SEARCH   */
          END;                                       /* END DO IF */

/***************************************/
/* LIST GENERIC EMPLOYEES              */
/***************************************/
        ELSE                         /* SELECT EMPLOYEES BY NAME  */
          DO;                        /* SEARCH ON PART OF NAME?   */
            IF INDEX(LNAMEWK,'%') > ZERO THEN
              DO;                      /* YES:  SEARCH ON PART OF */
                                       /* LAST NAME               */
                EXEC SQL OPEN TELE2;   /* OPEN CURSOR FOR SEARCH    */
                EXEC SQL FETCH TELE2 INTO :PPHONE;/* GET 1ST RECORD */

                IF SQLCODE = NOTFOUND THEN   /* NO RECORDS FOUND    */
                  DO;                         /* GET ERROR MESSAGE*/
                    CALL DSN8MPG (MODULE, '008I', OUTMSG);
                    PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
                  END;

                                    /* GET AND PRINT ALL RECORDS  */
                DO WHILE (SQLCODE = ZERO);
                  PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
                  EXEC SQL FETCH TELE2 INTO :PPHONE;/*GET NEXT RECORD*/
                END;                                  /* END DO WHILE */
                EXEC SQL CLOSE TELE2;  /* CLOSE CURSOR FOR SEARCH    */
              END;                                       /* END DO IF */

/***************************************/
/* LIST SPECIFIC EMPLOYEES             */
/***************************************/
            ELSE                    /* NO - SEARCH ON LAST NAME   */
              DO;                   /* & OPTIONALLY FIRST NAME    */
                                    /* SEE IF FIRST NAME ENTERED  */
                                    /* IF NOT SET UP FOR ALL NAMES*/
                EXEC SQL OPEN TELE3;  /* OPEN CURSOR FOR SEARCH     */
                EXEC SQL FETCH TELE3 INTO :PPHONE;/* GET 1ST RECORD */

                IF SQLCODE = NOTFOUND THEN       /* NO RECORDS FOUND*/
                  DO;                           /* GET ERROR MESSAGE*/
                    CALL DSN8MPG (MODULE, '008I', OUTMSG);
                    PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
                  END;

                                    /* GET AND PRINT ALL RECORDS  */
                DO WHILE (SQLCODE = ZERO);
                  PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
                  EXEC SQL FETCH TELE3 INTO :PPHONE;/*GET NEXT RECORD*/
                END;                                  /* END DO WHILE */
                EXEC SQL CLOSE TELE3;     /* CLOSE CURSOR FOR SEARCH*/
```

```
          END;                                 /* END DO ELSE */
        END;                                    /* END DO IF */
    END;                                         /*END WHEN */

/****************************************/
/* UPDATES PHONE NUMBERS FOR EMPLOYEES */
/****************************************/
    WHEN ('U') DO;                          /* TELEPHONE UPDATE */
      EXEC SQL UPDATE VEMPLP
                  SET PHONENUMBER = :NEWNO /* CHANGE PHONE NO.*/
                  WHERE EMPLOYEENUMBER = :ENO;

      IF SQLCODE = ZERO THEN                  /* WAS UPDATE OK? */
        DO;
          CALL DSN8MPG (MODULE, '004I', OUTMSG);   /* YES         */
          PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);/* YES      */
        END;                                     /*EMPLOYEE FOUND*/
                                               /*UPDATE SUCCESSFUL*/
      ELSE
        DO;
          CALL DSN8MPG (MODULE, '007E', OUTMSG);   /*UPDATE FAILED*/
          PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
        END;                                      /* END DO ELSE*/

    END;                                          /* END WHEN  */

    OTHERWISE                                /* INVALID REQUEST */
      DO;
        CALL DSN8MPG (MODULE, '068E', OUTMSG);
        PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
      END;                                     /* END OTHERWISE */
   END;                                           /* END SELECT*/

  GET FILE (CARDIN) EDIT (IOAREA) (R(L4));     /* READ NEXT REQUEST */
  END;                                             /* END  EOF */
  GOTO PGMEND;                       /* BYPASS SQL ERRORHANDLING   */

/****************************/
/* SQL ERROR CODE HANDLING */
/****************************/


   DCL
      DSNTIAR ENTRY OPTIONS(ASM,INTER,RETCODE);
   DCL
      DATA_LEN FIXED BIN(31) INIT(120);
   DCL
      DATA_DIM FIXED BIN(31) INIT(10);
   DCL
      1 ERROR_MESSAGE AUTOMATIC,
       3 ERROR_LEN    FIXED BIN(15) UNAL INIT((DATA_LEN*DATA_DIM)),
       3 ERROR_TEXT(DATA_DIM) CHAR(DATA_LEN);

/******************************************/
/* SQL ERROR OCCURRED - GET ERROR MESSAGE*/
/******************************************/
DBERROR:
                                                  /* SQL ERROR */
                                         /* PRINT ERROR MESSAGE*/
  CALL DSN8MPG (MODULE, '060E', OUTMSG);
  PUT FILE (REPORT)  EDIT (OUTMSG,SQLCODE) (SKIP(2),A,F(10));
  CALL DSNTIAR( SQLCA , ERROR_MESSAGE , DATA_LEN );

  IF PLIRETV = ZERO THEN             /*ZERO RETURN CODE FROM DSNTIAR*/
   DO I=ONE TO DIM(ERROR_TEXT,ONE);
     PUT FILE (REPORT) EDIT ( ERROR_TEXT(I)) (SKIP,A) ;
   END;

  ELSE
    DO;
       CALL DSN8MPG (MODULE, '075E', OUTMSG);
       PUT FILE (REPORT) EDIT    /*NON-ZERO RETURN CODE FROM DSNTIAR*/
                                 /*PRINT ERROR MESSAGE              */
             ( OUTMSG, PLIRETV )  ( SKIP(2), A, F(10)) ;
    END;

/***********************************************************/
/* SQL RETURN CODE HANDLING WHEN PROCESSING CANNOT PROCEED*/
/***********************************************************/

EXEC SQL WHENEVER SQLERROR   CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
```

```
 EXEC SQL WHENEVER NOT FOUND  CONTINUE;

 EXEC SQL ROLLBACK;                         /* PERFORM ROLLBACK    */

   IF SQLCODE = ZERO THEN
     DO;                                    /* ROLLBACK SUCCESSFUL,*/
                                            /* ALL UPDATES REMOVED */
       CALL DSN8MPG (MODULE, '053I', OUTMSG);
       PUT FILE (REPORT)  EDIT (OUTMSG) (SKIP(2),A);
     END;

   ELSE
     DO;                                        /* ROLLBACK FAILED,*/
                                                /* RETURN CODE IS: */
       CALL DSN8MPG (MODULE, '061E', OUTMSG);
       PUT FILE (REPORT)  EDIT (OUTMSG,SQLCODE) (SKIP(2),A,F(10));
     END;

 PGMEND:                                        /*  PROGRAM END    */
   END;
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8BF3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
      PROGRAM DSN8B3
**********************************************************************
*                                                                    *
*   MODULE NAME = DSN8BF3, PROGRAM DSN8B3                            *
*                                                                    *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                       *
*                       PHONE APPLICATION                            *
*                       BATCH                                        *
*                       FORTRAN                                      *
*                                                                    *
*      LICENSED MATERIALS - PROPERTY OF IBM                          *
*      5695-DB2                                                      *
*      (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.      *
*                                                                    *
*      STATUS = VERSION 4                                            *
*                                                                    *
*   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND          *
*              UPDATES THEM IF DESIRED.                              *
*                                                                    *
*   NOTES = NONE                                                     *
*                                                                    *
*                                                                    *
*   MODULE TYPE = FORTRAN PROGRAM                                    *
*      PROCESSOR = DB2  PRECOMPILER, VS FORTRAN                      *
*      MODULE SIZE = SEE LINK EDIT                                   *
*      ATTRIBUTES = NOT REENTRANT OR REUSABLE                        *
*                                                                    *
*   ENTRY POINT = DSN8BF3                                            *
*      PURPOSE  = SEE FUNCTION                                       *
*      LINKAGE  = INVOKED FROM DSN RUN                               *
*      INPUT    =                                                    *
*                                                                    *
*                 SYMBOLIC LABEL/NAME = FT05F001                     *
*                 DESCRIPTION = INPUT REQUEST FILE                   *
*                                                                    *
*                 SYMBOLIC LABEL/NAME = VPHONE                       *
*                 DESCRIPTION = VIEW OF TELEPHONE INFORMATION        *
*                                                                    *
*      OUTPUT   =                                                    *
*                                                                    *
*                 SYMBOLIC LABEL/NAME = FT06F001                     *
*                 DESCRIPTION = = PRINTED REPORT AND RESULTS         *
*                                                                    *
*                 SYMBOLIC LABEL/NAME = VEMPLP                       *
*                 DESCRIPTION = VIEW OF EMPLOYEE INFORMATION         *
*                                                                    *
*                                                                    *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION                    *
*                                                                    *
```

```
*   EXIT-ERROR =                                                    *
*                                                                   *
*      RETURN CODE = NONE                                           *
*                                                                   *
*      ABEND CODES =  NONE                                          *
*                                                                   *
*      ERROR-MESSAGES =                                             *
*            DSN8000I - REQUEST IS:  ...                            *
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED               *
*            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE    *
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE                  *
*            DSN8051I - PROGRAM ENDED                               *
*            DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED    *
*            DSN8060E - SQL ERROR, RETURN CODE IS:                  *
*            DSN8061E - ROLLBACK FAILED, RETURN CODE IS:            *
*            DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'       *
*            DSN8075E - MESSAGE FORMAT ERROR,                       *
*                       RETURN CODE IS:                             *
*                                                                   *
*   EXTERNAL REFERENCES =                                           *
*      ROUTINES/SERVICES =                                          *
*            DSNTIR   - TRANSLATE SQLCA INTO MESSAGES               *
*                                                                   *
*      DATA-AREAS = NONE                                            *
*                                                                   *
*      CONTROL-BLOCKS =                                             *
*         SQLCA               - SQL COMMUNICATION AREA              *
*                                                                   *
*   TABLES = NONE                                                   *
*                                                                   *
*   CHANGE-ACTIVITY =   NONE                                        *
*                                                                   *
*                                                                   *
*  *PSEUDOCODE*                                                     *
*                                                                   *
*  PROCEDURE                                                        *
*    DO WHILE MORE INPUT                                            *
*    GET INPUT                                                      *
*      CREATE REPORT HEADING                                        *
*      CASE (ACTION)                                                *
*                                                                   *
*        SUBCASE ('L')                                              *
*          IF LASTNAME IS '*' THEN                                  *
*            LIST ALL EMPLOYEES                                     *
*          ELSE                                                     *
*            IF LASTNAME CONTAINS '%' THEN                          *
*            LIST EMPLOYEES GENERIC                                 *
*            ELSE                                                   *
*              LIST EMPLOYEES SPECIFIC                              *
*        ENDSUB                                                     *
*                                                                   *
*        SUBCASE ('U')                                              *
*          UPDATE PHONENUMBER FOR EMPLOYEE                          *
*          WRITE CONFIRMATION MESSAGE                               *
*                                                                   *
*        OTHERWISE                                                  *
*          INVALID REQUEST                                          *
*        ENDSUB                                                     *
*                                                                   *
*      ENDCASE                                                      *
*      GET NEXT INPUT                                               *
*    END                                                            *
*                                                                   *
*    IF SQL ERROR OCCURS THEN                                       *
*      ROLLBACK                                                     *
*  END.                                                             *
*                                                                   *
*-------------------------------------------------------------------*

*************************************
*  SQL DECLARATION FOR VIEW VPHONE  *
*************************************

      EXEC SQL DECLARE VPHONE TABLE
     C   (LASTNAME       VARCHAR(15)  NOT NULL,
     C    FIRSTNAME      VARCHAR(12)  NOT NULL,
     C    MIDDLEINITIAL     CHAR( 1)  NOT NULL,
     C    PHONENUMBER       CHAR( 4)          ,
     C    EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
     C    DEPTNUMBER        CHAR( 3)  NOT NULL,
     C    DEPTNAME       VARCHAR(36)  NOT NULL)
```

```
*************************************
*  SQL DECLARATION FOR VIEW VEMPLP  *
*************************************

      EXEC SQL DECLARE VEMPLP TABLE
     C (EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
     C  PHONENUMBER       CHAR( 4)          )

*************************************
*  PPHONE FIELDS                    *
*************************************

      CHARACTER * 15 LASTNM
      CHARACTER * 12 FIRSTN
      CHARACTER *  1 MIDINI
      CHARACTER *  4 PHONEN
      CHARACTER *  6 EMPNO
      CHARACTER *  3 DEPTNO
      CHARACTER * 36 DEPTNM

*************************************
*  INPUT FIELDS                     *
*************************************

      CHARACTER * 1  ACTION
      CHARACTER * 15 LNAME
      CHARACTER * 12 FNAME
      CHARACTER *  6 ENO
      CHARACTER *  4 NEWNO
      CHARACTER * 15 LNAMEW
      CHARACTER * 12 FNAMEW

*************************************
*  SQL CURSORS                      *
*************************************

      EXEC SQL DECLARE TELE1 CURSOR FOR
     C    SELECT *
     C    FROM VPHONE

      EXEC SQL DECLARE TELE2 CURSOR FOR
     C    SELECT *
     C    FROM VPHONE
     C    WHERE  LASTNAME  LIKE :LNAMEW
     C      AND  FIRSTNAME LIKE :FNAMEW

      EXEC SQL DECLARE TELE3 CURSOR FOR
     C    SELECT *
     C    FROM VPHONE
     C    WHERE  LASTNAME    =  :LNAME
     C       AND FIRSTNAME LIKE :FNAMEW

*************************************
*  SQL RETURN CODES: OK/NOTFOUND    *
*************************************

      INTEGER OK/0/,NOTFND/100/

*************************************
*  REPORT FORMATS AND INPUT         *
*************************************

  100 FORMAT ('0',A29,A21,A28)
  200 FORMAT ('0',A9,7X,A10,3X,A7,1X,A5,2X,A8,
     C       1X,A4,1X,A4,/,38X,A6,1X,A6,3X,
     C       A4,1X,A4,1X,A4,/)
  300 FORMAT (' ',A15,1X,A12,4X,A1,4X,A4,3X,A6,3X,
     C       A3,2X,A36)
  400 FORMAT (A1,A15,A12,A6,A4)
  500 FORMAT ('0', A)
  600 FORMAT ('0', A, I8)
  700 FORMAT ('1',A,//,
     C       5X,'REQUEST',2X,'LAST NAME',8X,'FIRST NAME',4X,
     C       'EMPNO',3X,'NEW XT.NO',/,
     C       3X,'--',A1,6X,'--',A15,'--',A12,'--',A6,'--',A4,'     --')
  800 FORMAT ('1')

*************************************
*  MESSAGES                         *
*************************************

      CHARACTER * 30 DSN800
```

```
      CHARACTER * 48 DSN804
      CHARACTER * 60 DSN868
      CHARACTER * 59 DSN807
      CHARACTER * 45 DSN860
      CHARACTER * 59 DSN853
      CHARACTER * 51 DSN861
      CHARACTER * 45 DSN808
      CHARACTER * 64 DSN875
      CHARACTER * 32 DSN851

*************************************
*  VARIABLES USED WITH DSNTIR       *
*************************************

      INTEGER   ERRLEN /960/
      CHARACTER*120 ERRTXT(8)

*************************************
*  MISCELLANEOUS VARIABLES          *
*************************************

      INTEGER I, ICODE
      CHARACTER * 15 PERC15

*************************************
*  SQL COMMUNICATION AREA           *
*************************************

      EXEC SQL INCLUDE SQLCA

*************************************
*  DATA STATEMENTS                  *
*************************************

      DATA           PERC15
     C/'%%%%%%%%%%%%%%%'/

      DATA           DSN800
     C/'DSN8000I:  DSN8BF3-REQUEST IS:'/
      DATA           DSN804
     C/'DSN8004I:  DSN8BF3-EMPLOYEE SUCCESSFULLY UPDATED'/
      DATA           DSN868
     C/'DSN8068E:  DSN8BF3-INVALID REQUEST, SHOULD BE ''L'' OR ''U'''/
      DATA           DSN807
     C/'DSN8007E:  DSN8BF3-EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE'/
      DATA           DSN860
     C/'DSN8060E:  DSN8BF3-SQL ERROR, RETURN CODE IS:'/
      DATA           DSN853
     C/'DSN8053I:  DSN8BF3-ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED'/
      DATA           DSN861
     C/'DSN8061E:  DSN8BF3-ROLLBACK FAILED, SQLCODE IS:'/
      DATA           DSN808
     C/'DSN8008I:  DSN8BF3-NO EMPLOYEE FOUND IN TABLE'/
      DATA           DSN875
     C/'DSN8075E:  DSN8BF3-MESSAGE FORMAT ROUTINE ERROR, RETURN CODE IS:
     C'/
      DATA           DSN851
     C/'DSN8051I:  DSN8BF3-PROGRAM ENDED'/

*************************************
*  SQL RETURN CODE HANDLING         *
*************************************

      EXEC SQL WHENEVER SQLERROR   GOTO 4000
      EXEC SQL WHENEVER SQLWARNING GOTO 4000

*************************************
*  START OF PROGRAM                 *
*************************************


*************************************
* CONTINUE WHILE MORE INPUT         *
*************************************
1000  CONTINUE

*************************************
*   GET NEXT INPUT                  *
*************************************

      READ (UNIT=05,FMT=400,END=3000) ACTION, LNAME, FNAME, ENO, NEWNO
      WRITE (UNIT=06,FMT=700) DSN800, ACTION, LNAME, FNAME, ENO, NEWNO
```

```
      WRITE (UNIT=06,FMT=800)

*************************************
*   CREATE REPORT HEADING          *
*   SELECT ACTION                  *
*************************************

*                                        **CREATE REPORT HEADING
      WRITE (UNIT=06,FMT=100) '----------------------------',
     C                        ' TELEPHONE DIRECTORY ',
     C                        '----------------------------'
      WRITE (UNIT=06,FMT=200) 'LAST NAME', 'FIRST NAME', 'INITIAL',
     C    'PHONE', 'EMPLOYEE', 'WORK', 'WORK', 'NUMBER',
     C    'NUMBER', 'DEPT', 'DEPT', 'NAME'

*                                        **SELECT ACTION

*                                        **LIST EMPLOYEES
      IF (ACTION .EQ. 'L') THEN
          GOTO 1010
*                                        **PERFORM UPDATE
      ELSE IF (ACTION .EQ. 'U') THEN
               GOTO 1700
*                                        **INVALID REQUEST
           ELSE
               GOTO 1800
      END IF


1010  CONTINUE
************************************
*  ACTION - LIST                   *
************************************

          IF (LNAME .NE. '*') GOTO 1300

***************************************
*   LIST ALL EMPLOYEES                *
***************************************

*                                               **OPEN CURSOR
           EXEC SQL OPEN TELE1
           NBRETR = 0

 1100 CONTINUE

*                                               **GET EMPLOYEE
             EXEC SQL FETCH TELE1 INTO
     C                    :LASTNM,:FIRSTN,:MIDINI,
     C                    :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

             IF (SQLCOD .EQ. NOTFND) GO TO 1200

*                                               **LIST ALL EMPLOYEES
             NBRETR = NBRETR + 1
                 WRITE (UNIT=06,FMT=300)
     C                    LASTNM,FIRSTN,MIDINI,
     C                    PHONEN, EMPNO, DEPTNO, DEPTNM
           GO TO 1100

*                                               **NO EMPLOYEE FOUND
*                                               **PRINT ERROR MESSAGE
 1200 CONTINUE
            IF (NBRETR .EQ. 0)  WRITE (UNIT=06,FMT=500) DSN808

*                                               **CLOSE CURSOR
           EXEC SQL CLOSE TELE1
           GO TO 1000


*************************************
*        ELSE DETERMINE IF LASTNAME *
*        OR FIRSTNAME IS GIVEN      *
*************************************

1300      CONTINUE

              IPOS=INDEX(LNAME,'%')

********************************************
* REPLACE FIRST BLANK AND FOLLOWING        *
* CHARACTERS IN LASTNAME WORK (LNAMEW)     *
```

```
* WITH CHARACTER % FOR LIKE PREDICATE.     *
********************************************

       IBLANK=INDEX(LNAME,' ')
       IF (IBLANK .GT. 1 ) THEN
           LNAMEW = LNAME(1:IBLANK-1)//PERC15(1:15-IBLANK+1)
                           ELSE IF (IBLANK .EQ. 1) THEN
                   LNAMEW=PERC15
                   IPOS = 1
                                ELSE
       END IF


********************************************
* REPLACE FIRST BLANK AND FOLLOWING        *
* CHARACTERS IN FIRSTNAME WORK (FNAMEW)    *
* WITH CHARACTER % FOR LIKE PREDICATE.     *
********************************************

       IBLANK=INDEX(FNAME,' ')
       IF (IBLANK .GT. 1 ) THEN
           FNAMEW = FNAME(1:IBLANK-1)//PERC15(1:12-IBLANK+1)
                           ELSE IF (IBLANK .EQ. 1) THEN
                   FNAMEW=PERC15(1:12)
                                ELSE
       END IF

          IF (IPOS .LE. 0) GOTO 1600

********************************************
*     LIST GENERIC EMPLOYEES               *
********************************************
*                                               **OPEN CURSOR
             EXEC SQL OPEN TELE2
              NBRETR = 0

 1400        CONTINUE
*                                               **GET EMPLOYEES
               EXEC SQL FETCH TELE2 INTO
     C                    :LASTNM,:FIRSTN,:MIDINI,
     C                    :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

               IF (SQLCOD .EQ. NOTFND) GO TO 1500

*                                               **LIST GENERIC EMPLOYEES
                NBRETR = NBRETR + 1
               WRITE (UNIT=06,FMT=300)
     C                    LASTNM,FIRSTN,MIDINI,
     C                    PHONEN, EMPNO, DEPTNO, DEPTNM
               GOTO 1400

*                                               **EMPLOYEE NOT FOUND
*                                               **PRINT ERROR MESSAGE
 1500 CONTINUE
             IF (NBRETR .EQ. 0) WRITE (UNIT=06,FMT=500) DSN808

*                                               **CLOSE CURSOR
             EXEC SQL CLOSE TELE2
               GOTO 1000


********************************************
*    LIST SPECIFIC EMPLOYEES               *
********************************************

 1600 CONTINUE


*                                               **OPEN CURSOR
             EXEC SQL OPEN TELE3
              NBRETR = 0

 1620 CONTINUE
*                                               **GET EMPLOYEES
             EXEC SQL FETCH TELE3 INTO
     C                    :LASTNM,:FIRSTN,:MIDINI,
     C                    :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

           IF (SQLCOD .EQ. NOTFND) GO TO 1640

*                                              **LIST SPECIFIC EMPLOYEES
            NBRETR = NBRETR + 1
```

```
              WRITE (UNIT=06,FMT=300)
     C                     LASTNM,FIRSTN,MIDINI,
     C                     PHONEN, EMPNO, DEPTNO, DEPTNM
              GO TO 1620

*                                                **EMPLOYEE NOT FOUND
*                                                **PRINT ERROR MESSAGE
 1640 CONTINUE
             IF (NBRETR .EQ. 0) WRITE (UNIT=06,FMT=500) DSN808

*                                                **CLOSE CURSOR
            EXEC SQL CLOSE TELE3
            GO TO 1000


************************************************
*    UPDATE PHONE NUMBERS FOR EMPLOYEES        *
************************************************

1700  CONTINUE
*                                                **PERFORM UPDATE
          EXEC SQL UPDATE VEMPLP
     C                    SET PHONENUMBER = :NEWNO
     C                    WHERE EMPLOYEENUMBER = :ENO

          IF (SQLCOD .EQ. OK) THEN
*                                                **UPDATE SUCCESSFUL
*                                                **EMPLOYEE FOUND
*                                                **PRINT CONFIRMATION
*                                                **MESSAGE
              WRITE (UNIT=06,FMT=500) DSN804
                              ELSE
*                                                **UPDATE FAILED
*                                                **EMPLOYEE NOT FOUND
*                                                **PRINT ERROR MESSAGE
              WRITE (UNIT=06,FMT=500) DSN807
          END IF
          GO TO 1000

*                                                ** INVALID REQUEST
*                                                ** PRINT ERROR MESSAGE

 1800 CONTINUE
       WRITE (UNIT=06,FMT=500) DSN868
          GO TO 1000

******************
* END OF LOOP    *
* FOR MORE INPUT *
******************

*                                                **  THIS LABEL IS
*                                                **  BRANCHED TO FOR
*                                                **  END OF DATA
 3000 CONTINUE
       WRITE (UNIT=06,FMT=800)
       WRITE (UNIT=06,FMT=500) DSN851
       RETURN

*********************************************
*  IF SQL ERROR OCCURRED - GET ERROR MESSAGE*
*********************************************

       EXEC SQL WHENEVER SQLERROR   CONTINUE
       EXEC SQL WHENEVER SQLWARNING CONTINUE
       EXEC SQL WHENEVER NOT FOUND  CONTINUE

 4000 CONTINUE
*                                                **SQL ERROR
*                                                **PRINT ERROR MESSAGE
       WRITE (UNIT=06,FMT=600)  DSN860, SQLCOD
       CALL DSNTIR ( ERRLEN, ERRTXT, ICODE )

        IF (ICODE .EQ. OK)  THEN

            DO 4100 I=1, 10
            WRITE (UNIT=06,FMT=500) ERRTXT(I)
 4100       CONTINUE
                            ELSE
*                                                **ERROR DETECTED BY
*                                                **MESSAGE FORMAT
*                                                **ROUTINE
```

```
*                                               **PRINT ERROR MESSAGE

           WRITE (UNIT=06,FMT=600) DSN875, ICODE

       END IF
*                                               **PERFORM ROLLBACK
      EXEC SQL ROLLBACK
      IF (SQLCOD .EQ. OK) THEN

*                                               **ROLLBACK SUCCESSFUL
*                                               **PRINT CONFIRMATION
*                                               **MESSAGE
        WRITE (UNIT=06,FMT=500)  DSN853
                           ELSE

*                                               **ROLLBACK FAILED
*                                               **PRINT ERROR MESSAGE
        WRITE (UNIT=06,FMT=600)  DSN861, SQLCOD
      END IF
      RETURN
      END
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8HC3

THIS MODULE DISPLAYS THE Db2 DEPARTMENT AND EMPLOYEE TABLES AND UPDATES THEM IF DESIRED.

```
        IDENTIFICATION DIVISION.                                        00000100
       *-----------------------                                         00000200
        PROGRAM-ID. DSN8HC3.                                            00000300
                                                                        00000400
       *---------------------------------------------------------------* 00000500
       *                                                               * 00000600
       *   MODULE NAME = DSN8HC3                                       * 00000700
       *                                                               * 00000800
       *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                  * 00000900
       *                      ORGANIZATION APPLICATION                 * 00001000
       *                      ISPF                                     * 00001100
       *                      COBOL                                    * 00001200
       *  LICENSED MATERIALS - PROPERTY OF IBM                         * 00001300
       *  5615-DB2                                                     * 00001400
       *  (C) COPYRIGHT 1982, 2013 IBM CORP.  ALL RIGHTS RESERVED.     * 00001500
       *                                                               * 00001600
       *  STATUS = VERSION 11                                          * 00001700
       *                                                               * 00001800
       *                                                               * 00001900
       *                                                               * 00002000
       *   FUNCTION = THIS MODULE DISPLAYS THE DB2 DEPARTMENT AND      * 00002100
       *              EMPLOYEE TABLES AND UPDATES THEM IF DESIRED.     * 00002200
       *                                                               * 00002300
       *   NOTES =                                                     * 00002400
       *      DEPENDENCIES = REQUIRED ISPF PANELS:                     * 00002500
       *                     DSN8SSH                                   * 00002600
       *                     DSN8SSH1                                  * 00002700
       *                     DSN8SSH2                                  * 00002800
       *                     DSN8SSH3                                  * 00002900
       *                     DSN8SSH4                                  * 00003000
       *                     DSN8SSH5                                  * 00003100
       *      RESTRICTIONS = NONE                                      * 00003200
       *                                                               * 00003300
       *   MODULE TYPE = VS COBOL II PROGRAM                           * 00003400
       *      PROCESSOR   = DB2 PRECOMPILER, VS COBOL II               * 00003500
       *      MODULE SIZE = SEE LINKEDIT                               * 00003600
       *      ATTRIBUTES  = NOT REENTRANT OR REUSABLE                  * 00003700
       *                                                               * 00003800
       *   ENTRY POINT =  DSN8HC3                                      * 00003900
       *      PURPOSE = SEE FUNCTION                                   * 00004000
       *      LINKAGE = INVOKED FROM ISPF                              * 00004100
       *                                                               * 00004200
       *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:   * 00004300
       *              INPUT-MESSAGE:                                   * 00004400
       *                                                               * 00004500
       *                     SYMBOLIC LABEL/NAME = DSN8SSH             * 00004600
       *                     DESCRIPTION = MAIN MENU                   * 00004700
```

```
*                                                               * 00004800
*                     SYMBOLIC LABEL/NAME = DSN8SSH2            * 00004900
*                     DESCRIPTION = DEPARTMENT PANEL            * 00005000
*                                                               * 00005100
*                     SYMBOLIC LABEL/NAME = DSN8SSH3            * 00005200
*                     DESCRIPTION = SELECT FROM LIST PANEL      * 00005300
*                                                               * 00005400
*                     SYMBOLIC LABEL/NAME = DSN8SSH4            * 00005500
*                     DESCRIPTION = SELECT FROM LIST PANEL      * 00005600
*                                                               * 00005700
*                     SYMBOLIC LABEL/NAME = DSN8SSH5            * 00005800
*                     DESCRIPTION = EMPLOYEE PANEL              * 00005900
*                                                               * 00006000
*                     SYMBOLIC LABEL/NAME = VHDEPT              * 00006100
*                     DESCRIPTION = VIEW OF DEPARTMENT DATA     * 00006200
*                                                               * 00006300
*                     SYMBOLIC LABEL/NAME = VEMP                * 00006400
*                     DESCRIPTION = VIEW OF EMPLOYEE DATA       * 00006500
*                                                               * 00006600
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                 * 00006700
*               OUTPUT-MESSAGE:                                 * 00006800
*                                                               * 00006900
*                     SYMBOLIC LABEL/NAME = DSN8SSH             * 00007000
*                     DESCRIPTION = MAIN MENU PANEL             * 00007100
*                                                               * 00007200
*                     SYMBOLIC LABEL/NAME = DSN8SSH1            * 00007300
*                     DESCRIPTION = DEPARTMENT STRUCTURE PANEL  * 00007400
*                                                               * 00007500
*                     SYMBOLIC LABEL/NAME = DSN8SSH2            * 00007600
*                     DESCRIPTION = DEPARTMENT PANEL            * 00007700
*                                                               * 00007800
*                     SYMBOLIC LABEL/NAME = DSN8SSH3            * 00007900
*                     DESCRIPTION = SELECTION LIST PANEL        * 00008000
*                                                               * 00008100
*                     SYMBOLIC LABEL/NAME = DSN8SSH4            * 00008200
*                     DESCRIPTION = SELECTION LIST PANEL        * 00008300
*                                                               * 00008400
*                     SYMBOLIC LABEL/NAME = DSN8SSH5            * 00008500
*                     DESCRIPTION = EMPLOYEE PANEL              * 00008600
*                                                               * 00008700
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION               * 00008800
*                                                               * 00008900
*   EXIT-ERROR =                                                * 00009000
*                                                               * 00009100
*      RETURN CODE = NONE                                       * 00009200
*                                                               * 00009300
*      ABEND CODES =  NONE                                      * 00009400
*                                                               * 00009500
*                                                               * 00009600
*      ERROR-MESSAGES =                                         * 00009700
*            DSN8001I - EMPLOYEE NOT FOUND                      * 00009800
*            DSN8002I - EMPLOYEE SUCCESSFULLY ADDED             * 00009900
*            DSN8003I - EMPLOYEE SUCCESSFULLY ERASED            * 00010000
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED           * 00010100
*            DSN8005E - EMPLOYEE EXISTS ALREADY, ADD NOT DONE   * 00010200
*            DSN8006E - EMPLOYEE DOES NOT EXIST, ERASE NOT DONE * 00010300
*            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE* 00010400
*            DSN8011I - DEPARTMENT NOT FOUND                    * 00010500
*            DSN8012I - DEPARTMENT SUCCESSFULLY ADDED           * 00010600
*            DSN8013I - DEPARTMENT SUCCESSFULLY ERASED          * 00010700
*            DSN8014I - DEPARTMENT SUCCESSFULLY UPDATED         * 00010800
*            DSN8015E - DEPARTMENT EXISTS ALREADY, ADD NOT DONE * 00010900
*            DSN8016E - DEPARTMENT DOES NOT EXIST, ERASE NOT    * 00011000
*                       DONE                                    * 00011100
*            DSN8017E - DEPARTMENT DOES NOT EXIST, UPDATE NOT   * 00011200
*                       DONE                                    * 00011300
*            DSN8060E - SQL ERROR, RETURN CODE IS:              * 00011400
*            DSN8074E - DATA IS TOO LONG FOR SEARCH CRITERIA    * 00011500
*            DSN8079E - CONNECTION NOT ESTABLISHED              * 00011600
*            DSN8200E - INVALID DEPARTMENT NUMBER, EMPLOYEE NOT * 00011700
*                       ADDED                                   * 00011800
*            DSN8203E - INVALID WORK DEPT, EMPLOYEE NOT UPDATED * 00011900
*            DSN8210E - INVALID MGRNO, DEPARTMENT NOT ADDED     * 00012000
*            DSN8213E - INVALID ADMIN DEPT ID, DEPARTMENT NOT   * 00012100
*                       ADDED                                   * 00012200
*            DSN8214E - INVALID MANAGER ID, DEPARTMENT NOT      * 00012300
*                       UPDATED                                 * 00012400
*            DSN8215E - INVALID ADMIN DEPT ID, DEPARTMENT NOT   * 00012500
*                       UPDATED                                 * 00012600
*            DSN8216E - DEPT NOT AT SPECIFIED LOCATION, EMPLOYEE* 00012700
*                       NOT ADDED                               * 00012800
*            DSN8217E - DEPT NOT AT SPECIFIED LOCATION, EMP NOT * 00012900
```

```
*                      UPDATED                                 *
*                                                              *
*   EXTERNAL REFERENCES =                                      *
*      ROUTINES/SERVICES =                                     *
*         DSN8MCG             - ERROR MESSAGE ROUTINE          *
*         ISPLINK             - ISPF SERVICES ROUTINE          *
*                                                              *
*      DATA-AREAS =                                            *
*         NONE                                                 *
*                                                              *
*      CONTROL-BLOCKS =                                        *
*         SQLCA               - SQL COMMUNICATION AREA         *
*                                                              *
*   TABLES = NONE                                              *
*                                                              *
*                                                              *
*   CHANGE-ACTIVITY = NONE                                     *
*                                                              *
*  *PSEUDOCODE*                                                *
*                                                              *
*   SET UP RETURN CODE HANDLING               0000-PROGRAM-START*
*   SET PREVIOUS LOCATION TO LOCAL                             *
*   DO UNTIL NO MORE TERMINAL INPUT                            *
*      GET PANEL INPUT                        1000-MAIN-LOOP   *
*      IF CURRENT AND PREVIOUS LOCATION DIFFER THEN            *
*        IF REMOTE LOCATION THEN                               *
*            CONNECT TO REMOTE LOCATION                        *
*        ELSE RESET TO LOCAL LOCATION                          *
*      DETERMINE PROCESSING REQUEST                            *
*         IF ACTION FIELD ADD:                                 *
*            IF OBJECT FIELD IS DE:                            *
*               ADD RECORD TO VHDEPT TABLES   2000-ADDDEPT     *
*                  AT ALL LOCATIONS                            *
*            ELSE IF OBJECT FIELD IS EM:                       *
*               ADD RECORD TO VEMP TABLE      3000-ADDEMP      *
*            ELSE:                                             *
*         ELSE:                                                *
*            IF OBJECT FIELD DE OR DS:        5000-DEPARTMENT  *
*               IF "LIST GENERIC":            5100-GENDEPT     *
*                  CHOOSE CURSOR BASED ON     5110-GETDEPTTAB  *
*                     SEARCH CRITERIA AND DATA                 *
*                  CREATE ISPF TABLE                           *
*                  DO UNTIL NO MORE RECORDS:                   *
*                     FETCH RECORD                             *
*                     STORE RECORD IN TABLE                    *
*                  DISPLAY DEPARTMENT LIST    5121-GETDEPT     *
*                     ON SCREEN                                *
*                  STORE SELECTED DEPT ID IN                   *
*                     HOST VARIABLE                            *
*               ELSE:                                          *
*               IF OBJFLD IS DE:              5200-DISPLAYDEPT *
*                  FETCH SELECTED DEPT                         *
*                  DISPLAY DEPT ON SCREEN     5210-DISDEPTACT  *
*                  IF ACTION IS ERASE:        5220-ERASEDEPT   *
*                     DELETE DEPARTMENT AT    5221-DELDEPTS    *
*                        ALL LOCATIONS                         *
*                     PERFORM CASCADE DELETE  5223-DELDEPEND   *
*                        OF DEPENDENT DEPTS                    *
*                        AT ALL LOCATIONS                      *
*                  ELSE IF ACTION IS UPDATE:  5230-UPDATEDEPT  *
*                     UPDATE DEPARTMENT AT                     *
*                        ALL LOCATIONS                         *
*                  ELSE:                                       *
*               ELSE (OBJFLD IS DS):          5300-STRUCTURE   *
*                  FETCH SELECTED DEPT                         *
*                  DISPLAY SELECTED DEPT                       *
*                  CREATE ISPF TABLE          5310-DISSTR      *
*                  DO UNTIL NO MORE RECORDS:  5312-GETSTRTAB   *
*                     FETCH DEPT REPORTING TO                  *
*                        SELECTED DEPT                         *
*                     STORE RECORD IN TABLE                    *
*                  DISPLAY DEPT LIST ON SCREEN                 *
*            ELSE  (OBJFLD IS EM):            6000-EMPLOYEE    *
*               IF "LIST GENERIC":            6100-GENEMP      *
*                  CHOOSE CURSOR BASED ON     6110-GETEMPTAB   *
*                     SEARCH CRITERIA AND DATA                 *
*                  CREATE ISPF TABLE                           *
*                  DO UNTIL NO MORE RECORDS:                   *
*                     FETCH RECORD                             *
*                     STORE RECORD IN TABLE                    *
*                  DISPLAY DEPARTMENT LIST    6121-GETEMP      *
*                     ON SCREEN                                *
```

```
      *                  STORE SELECTED DEPT ID IN                   *
      *                     HOST VARIABLE                            *
      *               ELSE:                                          *
      *               FETCH SELECTED EMPLOYEE       6200-DISPLAYEMP   *
      *               DISPLAY EMPLOYEE ON SCREEN                     *
      *               IF ACTION IS ERASE:           6220-ERASEEMP     *
      *                  DELETE EMPLOYEE FROM VEMP                   *
      *               ELSE IF ACTION IS UPDATE:     6230-UPDATEEMP    *
      *                  UPDATE EMPLOYEE IN VEMP                     *
      *   END-DO UNTIL NO MORE TERMINAL INPUT                        *
      *   RELEASE ALL CONNECTIONS                                    *
      *--------------------------------------------------------------*
       ENVIRONMENT DIVISION.
      *------------------------
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MSGOUT ASSIGN TO UT-S-SYSPRINT.

       DATA DIVISION.
      *------------------------
       FILE SECTION.

       FD  MSGOUT
           RECORD CONTAINS 71 CHARACTERS
           LABEL RECORDS ARE OMITTED.
       01  MSGREC                  PIC X(71).

       WORKING-STORAGE SECTION.
      *--------------------------------------------------------------*
       77  COIBM                       PIC X(54) VALUE IS
           'COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1990'.
       77  MODULE                      PIC X(07)  VALUE 'DSN8HC3'.
       77  MSGS-VAR                    PIC X(08)  VALUE 'DSN8MSGS'.
       77  MSGCODE                     PIC X(06).
       77  SEL-EXIT                    PIC X(01).
       77  GEND-EXIT                   PIC X(01).
       77  GENE-EXIT                   PIC X(01).
       77  SPECIAL-EXIT                PIC X(01).
       77  ROWS-CHANGED                PIC 9(04).
       77  NUMROWS                     PIC 9(08).
       77  PERCENT-COUNTER             PIC S9(04) COMP.
       77  LENGTH-COUNTER              PIC S9(04) COMP.
       77  W-BLANK                     PIC X(01)  VALUE ' '.
      *-------------------------------------------------------------*
      * ISPF DIALOG VARIABLE NAMES                                  *
      *-------------------------------------------------------------*
           EXEC SQL INCLUDE SQLCA END-EXEC.
       01  LIST-PANEL-VARS.
           03  CH-VAR                  PIC X(08)  VALUE 'ZTDSELS '.
           03  QROWS                   PIC X(08)  VALUE 'QROWS   '.
      *
      *    ACTION PANEL VARIABLES
      *
           03  ACT-VAR                 PIC X(08)  VALUE 'A       '.
           03  OBJ-VAR                 PIC X(08)  VALUE 'OB      '.
           03  SEA-VAR                 PIC X(08)  VALUE 'SE      '.
           03  LOC-VAR                 PIC X(08)  VALUE 'LOCATION'.
           03  DAT-VAR                 PIC X(08)  VALUE 'NAMEID  '.
      *
      *    DEPARTMENT STRUCTURE VARIABLES
      *
           03  DN1M-VAR                PIC X(08)  VALUE 'MDEPIDP '.
           03  DNAME1M-VAR             PIC X(08)  VALUE 'MDEPNAMP'.
           03  DMGR1M-VAR              PIC X(08)  VALUE 'MMGRIDP '.
           03  EFN1M-VAR               PIC X(08)  VALUE 'MMGNAMP '.
           03  EMI1M-VAR               PIC X(08)  VALUE 'MMGMIP  '.
           03  ELN1M-VAR               PIC X(08)  VALUE 'MMGLNMP '.
           03  DN1-VAR                 PIC X(08)  VALUE 'DEPIDP  '.
           03  DNAME1-VAR              PIC X(08)  VALUE 'DEPNAMP '.
           03  DMGR1-VAR               PIC X(08)  VALUE 'MGRIDP  '.
           03  EFN1-VAR                PIC X(08)  VALUE 'MGNAMP  '.
           03  EMI1-VAR                PIC X(08)  VALUE 'MGMIP   '.
           03  ELN1-VAR                PIC X(08)  VALUE 'MGLNMP  '.
      *
      *    DISPLAY PANEL VARIABLES
      *
           03  ACTL-VAR                PIC X(08)  VALUE 'PACTION '.
           03  DN2-VAR                 PIC X(08)  VALUE 'DEPID2  '.
           03  DNAME2-VAR              PIC X(08)  VALUE 'DEPNAM2 '.
           03  DMGR2-VAR               PIC X(08)  VALUE 'MGRID2  '.
           03  DADM-VAR                PIC X(08)  VALUE 'MDEPID2 '.
           03  DLOC-VAR                PIC X(08)  VALUE 'DEPLOC2 '.
```

```
       03  EN2-VAR                PIC X(08)  VALUE 'EMPID2  '.
       03  EFN2-VAR               PIC X(08)  VALUE 'EMPNAM2 '.
       03  EMI2-VAR               PIC X(08)  VALUE 'EMPMI2  '.
       03  ELN2-VAR               PIC X(08)  VALUE 'MLNM2   '.
       03  EWD-VAR                PIC X(08)  VALUE 'DEPIDB2 '.
  *
  *    SELECT DEPARTMENT VARIABLES
  *
       03  SD-VAR                 PIC X(08)  VALUE 'SELECT  '.
       03  DN3-VAR                PIC X(08)  VALUE 'DID     '.
       03  DNAME3-VAR             PIC X(08)  VALUE 'DEPNGEN '.
       03  DMGR3-VAR              PIC X(08)  VALUE 'MID     '.
       03  MGRN-VAR               PIC X(08)  VALUE 'MNGEN   '.
  *
  *    SELECT EMPLOYEE VARIABLES
  *
       03  SEM-VAR                PIC X(08)  VALUE 'SELEC4  '.
       03  EN4-VAR                PIC X(08)  VALUE 'EMPID4  '.
       03  EMPN-VAR               PIC X(08)  VALUE 'EMPNM4  '.
       03  DN4-VAR                PIC X(08)  VALUE 'DEPID4  '.
       03  DNAME4-VAR             PIC X(08)  VALUE 'DPNAME4 '.
  *
  *    TABLE VARIABLES
  *
       03  DEPT-TABLE             PIC X(08)  VALUE 'DSN8DTAB'.
       03  DS-TABLE               PIC X(08)  VALUE 'DSN8STAB'.
       03  EMP-TABLE              PIC X(08)  VALUE 'DSN8ESEL'.
  *
  *    VARIABLE LISTS
  *
       03  ACTION-VARS            PIC X(27)  VALUE IS
       '( A OB SE LOCATION NAMEID )'.
       03  IDEN-VAR               PIC X(19)  VALUE IS
       '( PACTION )'.
       03  ADD-DEPT-VARS          PIC X(40)  VALUE IS
       '( DEPID2 DEPNAM2 MGRID2 MDEPID2 DEPLOC2)'.
       03  DEPT-VARS              PIC X(77)  VALUE IS
       '( DEPID2 DEPNAM2 MGRID2 MDEPID2 DEPLOC2 EMPID2 EMPNAM2 EMPMI
  -    '2 MLNM2 DEPIDB2 )'.
       03  ADD-EMP-VARS           PIC X(39)  VALUE IS
       '( EMPID2 EMPNAM2 EMPMI2 MLNM2 DEPIDB2 )'.
       03  SEL-EMP-VARS           PIC X(47)  VALUE IS
       '( ZTDSELS SELEC4 EMPID4 EMPNM4 DEPID4 DPNAME4 )'.
       03  SEL-DEPT-VARS          PIC X(40)  VALUE IS
       '( ZTDSELS SELECT DID DEPNGEN MID MNGEN )'.
       03  HEAD-DEPT-VARS         PIC X(51)  VALUE IS
       '( MDEPIDP MDEPNAMP MMGRIDP MMGNAMP MMGMIP MMGLNMP )'.
       03  DS-VARS                PIC X(45)  VALUE IS
       '( DEPIDP DEPNAMP MGRIDP MGNAMP MGMIP MGLNMP )'.
   01  PANEL-VARIABLE-LENGTHS.
       03  CH-VAR-STG             PIC 9(06)  COMP VALUE 04.
       03  QROWS-STG              PIC 9(06)  COMP VALUE 08.
  *
  *    ACTION PANEL VARIABLES
  *
       03  AC-VAR-STG             PIC 9(06)  COMP VALUE 01.
       03  OB-VAR-STG             PIC 9(06)  COMP VALUE 02.
       03  SE-VAR-STG             PIC 9(06)  COMP VALUE 02.
       03  LO-VAR-STG             PIC 9(06)  COMP VALUE 16.
       03  DT-VAR-STG             PIC 9(06)  COMP VALUE 36.
  *
  *    DEPARTMENT STRUCTURE VARIABLES
  *
       03  DN1M-VAR-STG           PIC 9(06)  COMP VALUE 03.
       03  DNAME1M-VAR-STG        PIC 9(06)  COMP VALUE 36.
       03  DMGR1M-VAR-STG         PIC 9(06)  COMP VALUE 06.
       03  EFN1M-VAR-STG          PIC 9(06)  COMP VALUE 12.
       03  EMI1M-VAR-STG          PIC 9(06)  COMP VALUE 01.
       03  ELN1M-VAR-STG          PIC 9(06)  COMP VALUE 15.
       03  DN1-VAR-STG            PIC 9(06)  COMP VALUE 03.
       03  DNAME1-VAR-STG         PIC 9(06)  COMP VALUE 36.
       03  DMGR1-VAR-STG          PIC 9(06)  COMP VALUE 06.
       03  EFN1-VAR-STG           PIC 9(06)  COMP VALUE 12.
       03  EMI1-VAR-STG           PIC 9(06)  COMP VALUE 01.
       03  ELN1-VAR-STG           PIC 9(06)  COMP VALUE 15.
  *
  *    DISPLAY PANEL VARIABLES
  *
       03  ACL-VAR-STG            PIC 9(06)  COMP VALUE 07.
       03  OCL-VAR-STG            PIC 9(06)  COMP VALUE 10.
       03  DN2-VAR-STG            PIC 9(06)  COMP VALUE 03.
       03  DNAME2-VAR-STG         PIC 9(06)  COMP VALUE 36.
```

```
           03  DMGR2-VAR-STG           PIC 9(06)  COMP VALUE 06.
           03  DADM-VAR-STG            PIC 9(06)  COMP VALUE 03.
           03  DLOC-VAR-STG            PIC 9(06)  COMP VALUE 16.
           03  EN2-VAR-STG             PIC 9(06)  COMP VALUE 06.
           03  EFN2-VAR-STG            PIC 9(06)  COMP VALUE 12.
           03  EMI2-VAR-STG            PIC 9(06)  COMP VALUE 01.
           03  ELN2-VAR-STG            PIC 9(06)  COMP VALUE 15.
           03  EWD-VAR-STG             PIC 9(06)  COMP VALUE 03.
      *
      *    SELECT DEPARTMENT VARIABLES
      *
           03  SD-VAR-STG              PIC 9(06)  COMP VALUE 01.
           03  DN3-VAR-STG             PIC 9(06)  COMP VALUE 03.
           03  DNAME3-VAR-STG          PIC 9(06)  COMP VALUE 36.
           03  DMGR3-VAR-STG           PIC 9(06)  COMP VALUE 06.
           03  MGRN-VAR-STG            PIC 9(06)  COMP VALUE 18.
      *
      *    SELECT EMPLOYEE VARIABLES
      *
           03  SEM-VAR-STG             PIC 9(06)  COMP VALUE 01.
           03  EN4-VAR-STG             PIC 9(06)  COMP VALUE 06.
           03  EMPN-VAR-STG            PIC 9(06)  COMP VALUE 17.
           03  DN4-VAR-STG             PIC 9(06)  COMP VALUE 03.
           03  DNAME4-VAR-STG          PIC 9(06)  COMP VALUE 36.
      *
           03  MSGS-VAR-STG            PIC 9(06)  COMP VALUE 79.
      *--------------------------------------------------------------*
      * ISPF DIALOG SERVICES DECLARATIONS                            *
      *--------------------------------------------------------------*
       01  I-VDEFINE               PIC X(08)  VALUE 'VDEFINE '.
       01  I-VGET                  PIC X(08)  VALUE 'VGET    '.
       01  I-VPUT                  PIC X(08)  VALUE 'VPUT    '.
       01  I-DISPLAY               PIC X(08)  VALUE 'DISPLAY '.
       01  I-TBDISPL               PIC X(08)  VALUE 'TBDISPL '.
       01  I-TBTOP                 PIC X(08)  VALUE 'TBTOP   '.
       01  I-TBCREATE              PIC X(08)  VALUE 'TBCREATE'.
       01  I-TBCLOSE               PIC X(08)  VALUE 'TBCLOSE '.
       01  I-TBADD                 PIC X(08)  VALUE 'TBADD   '.
       01  I-TBGET                 PIC X(08)  VALUE 'TBGET   '.
       01  I-TBQUERY               PIC X(08)  VALUE 'TBQUERY '.
      *--------------------------------------------------------------*
      * ISPF CALL MODIFIERS                                          *
      *--------------------------------------------------------------*
       01  I-NOWRITE               PIC X(08)  VALUE 'NOWRITE '.
       01  I-REPLACE               PIC X(08)  VALUE 'REPLACE '.
       01  I-CHAR                  PIC X(08)  VALUE 'CHAR    '.
      *--------------------------------------------------------------*
      * ISPF PANEL NAMES                                             *
      *--------------------------------------------------------------*
       01  SEL-PANEL               PIC X(08)  VALUE 'DSN8SSH '.
       01  STR-PANEL               PIC X(08)  VALUE 'DSN8SSH1'.
       01  DEPT-PANEL              PIC X(08)  VALUE 'DSN8SSH2'.
       01  GEND-PANEL              PIC X(08)  VALUE 'DSN8SSH3'.
       01  GENE-PANEL              PIC X(08)  VALUE 'DSN8SSH4'.
       01  EMP-PANEL               PIC X(08)  VALUE 'DSN8SSH5'.
      *--------------------------------------------------------------*
      * LOCAL-VARIABLES                                              *
      *--------------------------------------------------------------*
       01  LOCAL-VARIABLES.
           03  DATAW                       PIC X(36).
           03  GENDATA                     PIC X(36).
           03  SEL-DEPT                    PIC X(01).
           03  SEL-EMP                     PIC X(01).
           03  MGR-NAME                    PIC X(18).
           03  EMP-NAME                    PIC X(17).
           03  TOKEN                       PIC X(70).
           03  TEMPLOC                     PIC X(16).
           03  PREVLOC                     PIC X(16).
           03  TEMPDEPT                    PIC X(03).
           03  CURDEPT                     PIC X(03).
           03  DELDEPT                     PIC X(03).
           03  STACKTOP                    PIC S9(04).
           03  DEPTPTR                     PIC S9(04).
           03  LISTPTR                     PIC S9(04).
           03  LOCPTR                      PIC S9(04).
           03  LOCTOP                      PIC S9(04).
           03  CONVSQL                     PIC S9(15) COMP-3.
           03  OUTMSG                      PIC X(69).
           03  TMSG REDEFINES OUTMSG.
               05  TMSGTXT                 PIC X(46).
               05  FILLER                  PIC X(23).
           03  MSGS                        PIC X(79) VALUE SPACES.
```

```
     03  MSGS-DETAIL REDEFINES MSGS.
         05  OUT-MESSAGE             PIC X(46).
         05  SQL-CODE                PIC +(04).
         05  FILLER                  PIC X(29).

 01  CONV                   PIC S9(15) COMP-3.
 01  DEPTS-TABLE.
     03  DEPTS OCCURS 1000 TIMES.
         05  DEPTS-ITEM              PIC X(03).
 01  DEPTLIST-TABLE.
     03  DEPTLIST OCCURS 1000 TIMES.
         05  DEPTLIST-ITEM           PIC X(03).
 01  LOCLIST-TABLE.
     03  LOCLIST OCCURS 1000 TIMES.
         05  LOCLIST-ITEM            PIC X(16).
*-----------------------------------------------------------------*
* ACTION PANEL - IO AREA                                          *
*-----------------------------------------------------------------*
 01  PGM-PANEL-VARS.
     03  ACTION                  PIC X(01).
     03  OBJFLD                  PIC X(02).
     03  SEARCH-CRIT             PIC X(02).
     03  LOCATION                PIC X(16).
     03  NAMEID                  PIC X(36).
     03  ACTION-LIST             PIC X(07).
*-----------------------------------------------------------------*
* EMPLOYEE RECORD - IO AREA                                       *
*-----------------------------------------------------------------*
 01  EMP-RECORD.
     02  EMP-NUMB           PIC X(06).
     02  EMP-FIRST-NAME     PIC X(12).
     02  EMP-MID-INIT       PIC X(01).
     02  EMP-LAST-NAME      PIC X(15).
     02  EMP-WORK-DEPT      PIC X(03).
 01  EMP-INDICATOR-TABLE.
     02  WORK-DEPT-IND      PIC S9(4) COMP.
*-----------------------------------------------------------------*
* EMPLOYEE RECORD FOR DEPT STRUCTURE - IO AREA                    *
*-----------------------------------------------------------------*
 01  EMP1-RECORD.
     02  EMP1-NUMB          PIC X(06).
     02  EMP1-FIRST-NAME    PIC X(12).
     02  EMP1-MID-INIT      PIC X(01).
     02  EMP1-LAST-NAME     PIC X(15).
     02  EMP1-WORK-DEPT     PIC X(03).
 01  EMP1-INDICATOR-TABLE.
     02  WORK1-DEPT-IND     PIC S9(4) COMP.
*-----------------------------------------------------------------*
* DEPARTMENT RECORD - IO AREA                                     *
*-----------------------------------------------------------------*
 01  DEPT-RECORD.
     02  DEPT-NUMB          PIC X(03).
     02  DEPT-NAME          PIC X(36).
     02  DEPT-MGR           PIC X(06).
     02  DEPT-ADMR          PIC X(03).
     02  DEPT-LOC           PIC X(16).
 01  DEPT-INDICATOR-TABLE.
     02  DEPT-MGR-IND       PIC S9(4) COMP.
*-----------------------------------------------------------------*
* DEPARTMENT RECORD FOR DEPT STRUCTURE - IO AREA                  *
*-----------------------------------------------------------------*
 01  DEPT1-RECORD.
     02  DEPT1-NUMB         PIC X(03).
     02  DEPT1-NAME         PIC X(36).
     02  DEPT1-MGR          PIC X(06).
     02  DEPT1-ADMR         PIC X(03).
     02  DEPT1-LOC          PIC X(16).
 01  DEPT1-INDICATOR-TABLE.
     02  DEPT1-MGR-IND      PIC S9(4) COMP.
*-----------------------------------------------------------------*
* SQLCA OUTPUT                                                    *
*-----------------------------------------------------------------*

 01  SQLCA-LINE0.
     02  FILLER            PIC X(45)  VALUE
     'DSN8060E  DSN8HC3 SQL ERROR, RETURN CODE IS: '.
     02  SQLCODE-MSG       PIC +(16).
     02  FILLER            PIC X(11)  VALUE SPACES.

 01  SQLCA-LINE1.
     02  FILLER            PIC X(05) VALUE SPACES.
     02  SQLCAID-NAME      PIC X(13) VALUE 'SQLCAID    = '.
```

```
           02  SQLCAID-VALUE      PIC X(08).
           02  FILLER             PIC X(14) VALUE SPACES.
           02  SQLCABC-NAME       PIC X(13) VALUE 'SQLABC     = '.
           02  SQLCABC-VALUE      PIC Z(15).
           02  FILLER             PIC X(03) VALUE SPACES.

       01  SQLCA-LINE2.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLCODE-NAME       PIC X(13) VALUE 'SQLCODE    = '.
           02  SQLCODE-VALUE      PIC +(16).
           02  FILLER             PIC X(07) VALUE SPACES.
           02  SQLERRML-NAME      PIC X(13) VALUE 'SQLERRML   = '.
           02  SQLERRML-VALUE     PIC Z(15).
           02  FILLER             PIC X(03) VALUE SPACES.

       01  SQLCA-LINE3.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLERRMC-NAME      PIC X(13) VALUE 'SQLERRMC = '.
           02  FILLER             PIC X(53) VALUE SPACES.

       01  SQLCA-LINE4.
           02  FILLER             PIC X(01) VALUE SPACES.
           02  SQLERRMC-VALUE     PIC X(70).

       01  SQLCA-LINE5.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLERRP-NAME       PIC X(13) VALUE 'SQLERRP    = '.
           02  SQLERRP-VALUE      PIC X(08).
           02  FILLER             PIC X(14) VALUE SPACES.
           02  SQLERRD1-NAME      PIC X(13) VALUE 'SQLERRD(1) = '.
           02  SQLERRD1-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(03) VALUE SPACES.

       01  SQLCA-LINE6.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLERRD2-NAME      PIC X(13) VALUE 'SQLERRD(2) = '.
           02  SQLERRD2-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(07) VALUE SPACES.
           02  SQLERRD3-NAME      PIC X(13) VALUE 'SQLERRD(3) = '.
           02  SQLERRD3-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(03) VALUE SPACES.

       01  SQLCA-LINE7.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLERRD4-NAME      PIC X(13) VALUE 'SQLERRD(4) = '.
           02  SQLERRD4-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(07) VALUE SPACES.
           02  SQLERRD5-NAME      PIC X(13) VALUE 'SQLERRD(5) = '.
           02  SQLERRD5-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(03) VALUE SPACES.

       01  SQLCA-LINE8.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLERRD6-NAME      PIC X(13) VALUE 'SQLERRD(6) = '.
           02  SQLERRD6-VALUE     PIC Z(14)9.
           02  FILLER             PIC X(07) VALUE SPACES.
           02  SQLWARN0-NAME      PIC X(13) VALUE 'SQLWARN0   = '.
           02  SQLWARN0-VALUE     PIC X.
           02  FILLER             PIC X(17) VALUE SPACES.

       01  SQLCA-LINE9.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLWARN1-NAME      PIC X(13) VALUE 'SQLWARN1   = '.
           02  SQLWARN1-VALUE     PIC X.
           02  FILLER             PIC X(21) VALUE SPACES.
           02  SQLWARN2-NAME      PIC X(13) VALUE 'SQLWARN2   = '.
           02  SQLWARN2-VALUE     PIC X.
           02  FILLER             PIC X(17) VALUE SPACES.

       01  SQLCA-LINE10.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLWARN3-NAME      PIC X(13) VALUE 'SQLWARN3   = '.
           02  SQLWARN3-VALUE     PIC X.
           02  FILLER             PIC X(21) VALUE SPACES.
           02  SQLWARN4-NAME      PIC X(13) VALUE 'SQLWARN4   = '.
           02  SQLWARN4-VALUE     PIC X.
           02  FILLER             PIC X(17) VALUE SPACES.

       01  SQLCA-LINE11.
           02  FILLER             PIC X(05) VALUE SPACES.
           02  SQLWARN5-NAME      PIC X(13) VALUE 'SQLWARN5   = '.
           02  SQLWARN5-VALUE     PIC X.
```

```
     02  FILLER            PIC X(21) VALUE SPACES.
     02  SQLWARN6-NAME     PIC X(13) VALUE 'SQLWARN6   = '.
     02  SQLWARN6-VALUE    PIC X.
     02  FILLER            PIC X(17) VALUE SPACES.

 01  SQLCA-LINE12.
     02  FILLER            PIC X(05) VALUE SPACES.
     02  SQLWARN7-NAME     PIC X(13) VALUE 'SQLWARN7   = '.
     02  SQLWARN7-VALUE    PIC X.
     02  FILLER            PIC X(21) VALUE SPACES.
     02  SQLWARN8-NAME     PIC X(13) VALUE 'SQLWARN8   = '.
     02  SQLWARN8-VALUE    PIC X.
     02  FILLER            PIC X(17) VALUE SPACES.

 01  SQLCA-LINE13.
     02  FILLER            PIC X(05) VALUE SPACES.
     02  SQLWARN9-NAME     PIC X(13) VALUE 'SQLWARN9   = '.
     02  SQLWARN9-VALUE    PIC X.
     02  FILLER            PIC X(21) VALUE SPACES.
     02  SQLWARNA-NAME     PIC X(13) VALUE 'SQLWARNA   = '.
     02  SQLWARNA-VALUE    PIC X.
     02  FILLER            PIC X(17) VALUE SPACES.

 01  SQLCA-LINE14.
     02  FILLER            PIC X(05) VALUE SPACES.
     02  SQLSTATE-NAME     PIC X(13) VALUE 'SQLSTATE   = '.
     02  SQLSTATE-VALUE    PIC X(05).
     02  FILLER            PIC X(48) VALUE SPACES.

****************************************************************
* LINKAGE SECTION                                              *
****************************************************************

 LINKAGE SECTION.

*---------------------------------------------------------------*
* SQL DECLARATION FOR VIEW VHDEPT                               *
*---------------------------------------------------------------*
     EXEC SQL DECLARE VHDEPT TABLE
         (DEPTNO   CHAR(3)     NOT NULL,
          DEPTNAME VARCHAR(36) NOT NULL,
          MGRNO    CHAR(6)             ,
          ADMRDEPT CHAR(3)     NOT NULL,
          LOCATION CHAR(16)) END-EXEC.
*---------------------------------------------------------------*
* SQL DECLARATION FOR VIEW VEMP                                 *
*---------------------------------------------------------------*
     EXEC SQL DECLARE VEMP TABLE
         (EMPNO     CHAR(6)        NOT NULL,
          FIRSTNME   VARCHAR(12)   NOT NULL,
          MIDINIT   CHAR(1)        NOT NULL,
          LASTNAME  VARCHAR(15)    NOT NULL,
          WORKDEPT  CHAR(3)) END-EXEC.
*---------------------------------------------------------------*
* SQL CURSORS                                                   *
*---------------------------------------------------------------*
*
     EXEC SQL DECLARE CURDEPTLOC CURSOR FOR
         SELECT LOCATION
         FROM VHDEPT
         WHERE DEPTNO = :EMP-WORK-DEPT
           AND LOCATION = CURRENT SERVER
     END-EXEC.
*
     EXEC SQL DECLARE DEPTLOC CURSOR FOR
         SELECT LOCATION
         FROM VHDEPT
         WHERE DEPTNO = :EMP-WORK-DEPT
     END-EXEC.
*
     EXEC SQL DECLARE LOCS CURSOR FOR
         SELECT DISTINCT LOCATION
         FROM VHDEPT
         WHERE LOCATION <> :LOCATION
           AND LOCATION <> '                '
           AND LOCATION <> CURRENT SERVER
     END-EXEC.
*
     EXEC SQL DECLARE SUBDEPTS CURSOR FOR
         SELECT DEPTNO
         FROM VHDEPT
         WHERE ADMRDEPT = :CURDEPT
```

```
             AND DEPTNO <> :CURDEPT
       END-EXEC.
      *
       EXEC SQL DECLARE DEPT1 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
             EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
           FROM VHDEPT, VEMP
           WHERE DEPTNO = :DATAW
             AND MGRNO = EMPNO
           UNION
           SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
             ' ', ' ', ' ', ' ', ' '
           FROM VHDEPT
           WHERE DEPTNO = :DATAW
             AND MGRNO IS NULL
       END-EXEC.
      *
       EXEC SQL DECLARE ALLDEPT1 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO,
             SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
           FROM VHDEPT, VEMP
           WHERE MGRNO = EMPNO
             AND DEPTNO LIKE :GENDATA
           UNION
           SELECT DEPTNO, DEPTNAME, MGRNO, ' '
           FROM VHDEPT
           WHERE MGRNO IS NULL
             AND DEPTNO LIKE :GENDATA
           ORDER BY 1
       END-EXEC.
      *
       EXEC SQL DECLARE ALLDEPT2 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO,
             SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
           FROM VHDEPT, VEMP
           WHERE MGRNO = EMPNO
             AND DEPTNAME LIKE :GENDATA
           UNION
           SELECT DEPTNO, DEPTNAME, MGRNO, ' '
           FROM VHDEPT
           WHERE MGRNO IS NULL
             AND DEPTNAME LIKE :GENDATA
           ORDER BY 1
       END-EXEC.
      *
       EXEC SQL DECLARE ALLDEPT3 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO,
             SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
           FROM VHDEPT, VEMP
           WHERE MGRNO = EMPNO
             AND MGRNO LIKE :GENDATA
           UNION
           SELECT DEPTNO, DEPTNAME, MGRNO, ' '
           FROM VHDEPT
           WHERE MGRNO IS NULL
             AND MGRNO LIKE :GENDATA
           ORDER BY 1
       END-EXEC.
      *
       EXEC SQL DECLARE ALLDEPT4 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO,
             SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
           FROM VHDEPT, VEMP
           WHERE MGRNO = EMPNO
             AND LASTNAME LIKE :GENDATA
           ORDER BY 1
       END-EXEC.
      *
       EXEC SQL DECLARE ALLDEPT5 CURSOR FOR
           SELECT DEPTNO, DEPTNAME, MGRNO,
             SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
           FROM VHDEPT, VEMP
           WHERE MGRNO = EMPNO
             AND DEPTNAME = :GENDATA
           UNION
           SELECT DEPTNO, DEPTNAME, MGRNO, ' '
           FROM VHDEPT
           WHERE MGRNO IS NULL
             AND DEPTNAME = :GENDATA
           ORDER BY 1
       END-EXEC.
      *
```

```
           EXEC SQL DECLARE ALLDEPT6 CURSOR FOR
               SELECT DEPTNO, DEPTNAME, MGRNO,
                 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
               FROM VHDEPT, VEMP
               WHERE MGRNO = EMPNO
                 AND MGRNO = :GENDATA
               UNION
               SELECT DEPTNO, DEPTNAME, MGRNO, ' '
               FROM VHDEPT
               WHERE MGRNO IS NULL
                 AND MGRNO = :GENDATA
               ORDER BY 1
           END-EXEC.
      *
           EXEC SQL DECLARE ALLDEPT7 CURSOR FOR
               SELECT DEPTNO, DEPTNAME, MGRNO,
                 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
               FROM VHDEPT, VEMP
               WHERE MGRNO = EMPNO
                 AND LASTNAME = :GENDATA
               ORDER BY 1
           END-EXEC.
      *
           EXEC SQL DECLARE EMP1 CURSOR FOR
               SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
                 EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
               FROM VHDEPT, VEMP
               WHERE EMPNO = :DATAW
                 AND WORKDEPT = DEPTNO
               UNION
               SELECT ' ', ' ', ' ', ' ', ' ',
                 EMPNO, FIRSTNME, MIDINIT, LASTNAME, ' '
               FROM VEMP
               WHERE EMPNO = :DATAW
                 AND WORKDEPT IS NULL
           END-EXEC.
      *
           EXEC SQL DECLARE ALLEMP1 CURSOR FOR
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, DEPTNAME
               FROM VHDEPT, VEMP
               WHERE DEPTNO = WORKDEPT
                 AND EMPNO LIKE :GENDATA
               UNION
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, ' '
               FROM VEMP
               WHERE WORKDEPT IS NULL
                 AND EMPNO LIKE :GENDATA
               ORDER BY 1
           END-EXEC.
      *
           EXEC SQL DECLARE ALLEMP2 CURSOR FOR
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, DEPTNAME
               FROM VHDEPT, VEMP
               WHERE DEPTNO = WORKDEPT
                 AND LASTNAME LIKE :GENDATA
               UNION
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, ' '
               FROM VEMP
               WHERE WORKDEPT IS NULL
                 AND LASTNAME LIKE :GENDATA
               ORDER BY 1
           END-EXEC.
      *
           EXEC SQL DECLARE ALLEMP3 CURSOR FOR
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, DEPTNAME
               FROM VHDEPT, VEMP
               WHERE DEPTNO = WORKDEPT
                 AND LASTNAME = :GENDATA
               UNION
               SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME,
                 WORKDEPT, ' '
               FROM VEMP
               WHERE WORKDEPT IS NULL
                 AND LASTNAME = :GENDATA
               ORDER BY 1
           END-EXEC.
      *
```

```
           EXEC SQL DECLARE DEPTSTR CURSOR FOR
               SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
                 FIRSTNME, MIDINIT, LASTNAME
               FROM VHDEPT, VEMP
               WHERE ADMRDEPT = :DATAW
                 AND MGRNO = EMPNO
               UNION
               SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
                 ' ', ' ', ' '
               FROM VHDEPT
               WHERE ADMRDEPT = :DATAW
                 AND MGRNO IS NULL
               ORDER BY 1
           END-EXEC.
      *
           EJECT
       PROCEDURE DIVISION.
      *---------------------------------------------------------------*
      * SQL RETURN CODE HANDLING                                      *
      *---------------------------------------------------------------*
           EXEC SQL WHENEVER SQLERROR   GOTO L8000-P3-DBERROR END-EXEC.
           EXEC SQL WHENEVER SQLWARNING GOTO L8000-P3-DBERROR END-EXEC.
           EXEC SQL WHENEVER NOT FOUND  CONTINUE END-EXEC.
      *
      *---------------------------------------------------------------*
      * DEFINE COBOL - SPF VARIABLES                                  *
      *---------------------------------------------------------------*
       0000-PROGRAM-START.
           CALL 'ISPLINK' USING I-VDEFINE, CH-VAR, ROWS-CHANGED,
                        I-CHAR, CH-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, QROWS, NUMROWS,
                        I-CHAR, QROWS-STG.
      *
      *    ACTION PANEL
      *
           CALL 'ISPLINK' USING I-VDEFINE, ACT-VAR, ACTION,
                        I-CHAR, AC-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, OBJ-VAR, OBJFLD,
                        I-CHAR, OB-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, SEA-VAR, SEARCH-CRIT,
                        I-CHAR, SE-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, LOC-VAR, LOCATION,
                        I-CHAR, LO-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DAT-VAR, NAMEID,
                        I-CHAR, DT-VAR-STG.
      *
      *    DEPARTMENT STRUCTURE PANEL
      *
           CALL 'ISPLINK' USING I-VDEFINE, DN1M-VAR, DEPT1-NUMB,
                        I-CHAR, DN1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DNAME1M-VAR, DEPT1-NAME,
                        I-CHAR, DNAME1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DMGR1M-VAR, DEPT1-MGR,
                        I-CHAR, DMGR1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EFN1M-VAR, EMP1-FIRST-NAME,
                        I-CHAR, EFN1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EMI1M-VAR, EMP1-MID-INIT,
                        I-CHAR, EMI1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, ELN1M-VAR, EMP1-LAST-NAME,
                        I-CHAR, ELN1M-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DN1-VAR, DEPT-NUMB,
                        I-CHAR, DN1-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DNAME1-VAR, DEPT-NAME,
                        I-CHAR, DNAME1-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DMGR1-VAR, DEPT-MGR,
                        I-CHAR, DMGR1-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EFN1-VAR, EMP-FIRST-NAME,
                        I-CHAR, EFN1-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EMI1-VAR, EMP-MID-INIT,
                        I-CHAR, EMI1-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, ELN1-VAR, EMP-LAST-NAME,
                        I-CHAR, ELN1-VAR-STG.
      *
      *    DISPLAY PANEL
      *
           CALL 'ISPLINK' USING I-VDEFINE, ACTL-VAR, ACTION-LIST,
                        I-CHAR, ACL-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DN2-VAR, DEPT-NUMB,
                        I-CHAR, DN2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DNAME2-VAR, DEPT-NAME,
                        I-CHAR, DNAME2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DMGR2-VAR, DEPT-MGR,
```

```
                        I-CHAR, DMGR2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DADM-VAR, DEPT-ADMR,
                        I-CHAR, DADM-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DLOC-VAR, DEPT-LOC,
                        I-CHAR, DLOC-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EN2-VAR, EMP-NUMB,
                        I-CHAR, EN2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EFN2-VAR, EMP-FIRST-NAME,
                        I-CHAR, EFN2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EMI2-VAR, EMP-MID-INIT,
                        I-CHAR, EMI2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, ELN2-VAR, EMP-LAST-NAME,
                        I-CHAR, ELN2-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EWD-VAR, EMP-WORK-DEPT,
                        I-CHAR, EWD-VAR-STG.
      *
      *    SELECT DEPARTMENT PANEL
      *
           CALL 'ISPLINK' USING I-VDEFINE, SD-VAR, SEL-DEPT,
                        I-CHAR, SD-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DN3-VAR, DEPT-NUMB,
                        I-CHAR, DN3-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DNAME3-VAR, DEPT-NAME,
                        I-CHAR, DNAME3-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DMGR3-VAR, DEPT-MGR,
                        I-CHAR, DMGR3-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, MGRN-VAR, MGR-NAME,
                        I-CHAR, MGRN-VAR-STG.
      *
      *    SELECT EMPLOYEE PANEL
      *
           CALL 'ISPLINK' USING I-VDEFINE, SEM-VAR, SEL-EMP,
                        I-CHAR, SEM-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EN4-VAR, EMP-NUMB,
                        I-CHAR, EN4-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, EMPN-VAR, EMP-NAME,
                        I-CHAR, EMPN-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DN4-VAR, EMP-WORK-DEPT,
                        I-CHAR, DN4-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, DNAME4-VAR, DEPT-NAME,
                        I-CHAR, DNAME4-VAR-STG.
      *
           CALL 'ISPLINK' USING I-VDEFINE, MSGS-VAR, MSGS,
                        I-CHAR, MSGS-VAR-STG.
      *
      *----------------------------------------------------------------*
      * MAIN PROGRAM                                                   *
      *----------------------------------------------------------------*
           MOVE 'N' TO SEL-EXIT.
           MOVE SPACES TO PREVLOC.
           PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT
               UNTIL SEL-EXIT = 'Y'.
           MOVE 0 TO RETURN-CODE.
           MOVE SPACES TO MSGS.
           CALL 'ISPLINK' USING I-VPUT, MSGS-VAR.
           GOBACK.
      *
       1000-MAIN-LOOP.
           CALL 'ISPLINK' USING I-DISPLAY, SEL-PANEL.
           IF RETURN-CODE = 8 THEN
               EXEC SQL COMMIT END-EXEC
               EXEC SQL RELEASE ALL SQL END-EXEC
               MOVE 'Y' TO SEL-EXIT
           ELSE
               MOVE SPACES TO MSGS
               MOVE 'N' TO GEND-EXIT, GENE-EXIT
               CALL 'ISPLINK' USING I-VGET, ACTION-VARS
               MOVE NAMEID TO DATAW
               MOVE 0 TO LENGTH-COUNTER
               INSPECT DATAW
                   TALLYING LENGTH-COUNTER FOR CHARACTERS
                   BEFORE INITIAL SPACE
               IF SEARCH-CRIT = 'DI' AND LENGTH-COUNTER > 3 OR
                   SEARCH-CRIT = 'MI' AND LENGTH-COUNTER > 6 OR
                   SEARCH-CRIT = 'EI' AND LENGTH-COUNTER > 6 THEN
                   MOVE '074E' TO MSGCODE
                   CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                   MOVE OUTMSG TO MSGS
               ELSE
                   PERFORM 1100-CONNECT THRU 1100-CONNECT-EXIT
                   PERFORM 1200-DOACTION THRU 1200-DOACTION-EXIT.
       1000-MAIN-LOOP-EXIT.
```

```
           EXIT.                                                        00103200
      *                                                                 00103300
      *---------------------------------------------------------------* 00103400
      * CONNECT TO NEW LOCATION                                       * 00103500
      *---------------------------------------------------------------* 00103600
       1100-CONNECT.                                                    00103700
           IF LOCATION NOT EQUAL TO PREVLOC THEN                        00103800
               MOVE LOCATION TO PREVLOC                                 00103900
               IF LOCATION NOT EQUAL TO SPACES THEN                     00104000
                   EXEC SQL CONNECT TO :LOCATION END-EXEC               00104100
               ELSE                                                     00104200
                   EXEC SQL CONNECT RESET END-EXEC.                     00104300
       1100-CONNECT-EXIT.                                               00104400
           EXIT.                                                        00104500
      *                                                                 00104600
      *---------------------------------------------------------------* 00104700
      * DETERMINE PROCESSING REQUEST                                  * 00104800
      *---------------------------------------------------------------* 00104900
       1200-DOACTION.                                                   00105000
           IF ACTION = 'A' THEN                                         00105100
               MOVE '    ADD' TO ACTION-LIST                            00105200
               IF OBJFLD = 'DE' THEN                                    00105300
                   PERFORM 2000-ADDDEPT THRU 2000-ADDDEPT-EXIT          00105400
               ELSE                                                     00105500
                   PERFORM 3000-ADDEMP THRU 3000-ADDEMP-EXIT            00105600
           ELSE                                                         00105700
               PERFORM 4000-ACTION THRU 4000-ACTION-EXIT                00105800
               IF OBJFLD = 'DE' OR OBJFLD = 'DS' THEN                   00105900
                   PERFORM 5000-DEPARTMENT THRU 5000-DEPARTMENT-EXIT    00106000
               ELSE                                                     00106100
                   PERFORM 6000-EMPLOYEE THRU 6000-EMPLOYEE-EXIT.       00106200
       1200-DOACTION-EXIT.                                              00106300
           EXIT.                                                        00106400
      *                                                                 00106500
      *---------------------------------------------------------------* 00106600
      * ADD A DEPARTMENT                                              * 00106700
      *---------------------------------------------------------------* 00106800
       2000-ADDDEPT.                                                    00106900
           CALL 'ISPLINK' USING I-VPUT, IDEN-VAR.                       00107000
           PERFORM 2100-DISDEPTDATA THRU 2100-DISDEPTDATA-EXIT.         00107100
           CALL 'ISPLINK' USING I-VPUT, ADD-DEPT-VARS.                  00107200
           CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL.                  00107300
           IF RETURN-CODE NOT EQUAL TO 8 THEN                           00107400
               EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC             00107500
               MOVE SPACES TO SQLERRP                                   00107600
               EXEC SQL INSERT INTO VHDEPT                              00107700
                        VALUES (:DEPT-NUMB, :DEPT-NAME, :DEPT-MGR,      00107800
                                :DEPT-ADMR, :DEPT-LOC)                  00107900
               END-EXEC                                                 00108000
               PERFORM 2200-ADDDEPTCODES THRU 2200-ADDDEPTCODES-EXIT    00108100
               EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR END-EXEC00108200
               PERFORM 2300-GETEMPREC THRU 2300-GETEMPREC-EXIT          00108300
               CALL 'ISPLINK' USING I-VPUT, DEPT-VARS                   00108400
               CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL.              00108500
       2000-ADDDEPT-EXIT.                                               00108600
           EXIT.                                                        00108700
      *                                                                 00108800
      *---------------------------------------------------------------* 00108900
      * DISPLAY INPUT DATA ON PANEL                                   * 00109000
      *---------------------------------------------------------------* 00109100
       2100-DISDEPTDATA.                                                00109200
           MOVE SPACES TO DEPT-RECORD.                                  00109300
           MOVE SPACES TO EMP-RECORD.                                   00109400
           IF SEARCH-CRIT = 'DI' THEN                                   00109500
               MOVE DATAW TO DEPT-NUMB                                  00109600
           ELSE                                                         00109700
               IF SEARCH-CRIT = 'DN' THEN                               00109800
                   MOVE DATAW TO DEPT-NAME                              00109900
               ELSE                                                     00110000
                   IF SEARCH-CRIT = 'MI' THEN                           00110100
                       MOVE DATAW TO DEPT-MGR.                          00110200
       2100-DISDEPTDATA-EXIT.                                           00110300
           EXIT.                                                        00110400
      *                                                                 00110500
      *---------------------------------------------------------------* 00110600
      * CHECK RETURN CODE FROM INSERT.  IF OK, ADD TO OTHER LOCATIONS.* 00110700
      *---------------------------------------------------------------* 00110800
       2200-ADDDEPTCODES.                                               00110900
           IF SQLERRP = SPACES THEN                                     00111000
               MOVE '079E' TO MSGCODE                                   00111100
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG             00111200
               MOVE OUTMSG TO MSGS                                      00111300
```

```
     ELSE
         IF SQLCODE = -803 THEN
             MOVE '015E' TO MSGCODE
             CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
             MOVE OUTMSG TO MSGS
         ELSE
             IF SQLCODE = -530 THEN
                 UNSTRING SQLERRMC
                     DELIMITED BY HIGH-VALUE
                     INTO TOKEN
                 IF TOKEN = 'RDD' THEN
                     MOVE '213E' TO MSGCODE
                     CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                     MOVE OUTMSG TO MSGS
                 ELSE
                     IF TOKEN = 'RDE' THEN
                         MOVE '210E' TO MSGCODE
                         CALL 'DSN8MCG' USING MODULE, MSGCODE,
                             OUTMSG
                         MOVE OUTMSG TO MSGS
                     ELSE
                         GO TO L8000-P3-DBERROR
             ELSE
                 IF SQLCODE NOT EQUAL TO 0 THEN
                     GO TO L8000-P3-DBERROR
                 ELSE
                     EXEC SQL OPEN LOCS END-EXEC
                     MOVE 0 TO LOCPTR
                     PERFORM 2210-BUILDLOCTABLE THRU
                             2210-BUILDLOCTABLE-EXIT
                             UNTIL SQLCODE NOT EQUAL TO 0
                     EXEC SQL CLOSE LOCS END-EXEC
                     MOVE LOCPTR TO LOCTOP
                     MOVE 0 TO LOCPTR
                     PERFORM 2220-ADDLOCS THRU 2220-ADDLOCS-EXIT
                         UNTIL LOCPTR = LOCTOP
                         MOVE '012I' TO MSGCODE
                         CALL 'DSN8MCG' USING MODULE, MSGCODE,
                             OUTMSG
                         MOVE OUTMSG TO MSGS
                         MOVE DEPT-LOC TO LOCATION
                         PERFORM 1100-CONNECT THRU
                                 1100-CONNECT-EXIT.
 2200-ADDDEPTCODES-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* BUILD TABLE OF UNIQUE LOCATIONS IN VHDEPT                     *
*---------------------------------------------------------------*
 2210-BUILDLOCTABLE.
     EXEC SQL FETCH LOCS INTO :TEMPLOC END-EXEC.
     IF SQLCODE = 0 THEN
         ADD 1 TO LOCPTR
         MOVE TEMPLOC TO LOCLIST (LOCPTR).
 2210-BUILDLOCTABLE-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* ADD NEW DEPARTMENT TO VHDEPT VIEWS AT ALL LOCATIONS           *
*---------------------------------------------------------------*
 2220-ADDLOCS.
     IF LOCPTR < LOCTOP THEN
         ADD 1 TO LOCPTR
         MOVE LOCLIST (LOCPTR) TO TEMPLOC
         EXEC SQL CONNECT TO :TEMPLOC END-EXEC
         EXEC SQL INSERT INTO VHDEPT
                  VALUES (:DEPT-NUMB, :DEPT-NAME, :DEPT-MGR,
                          :DEPT-ADMR, :DEPT-LOC)
         END-EXEC.
 2220-ADDLOCS-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* RETRIEVE MANAGER INFO FOR NEW DEPARTMENT                      *
*---------------------------------------------------------------*
 2300-GETEMPREC.
     CALL 'ISPLINK' USING I-VGET, ADD-DEPT-VARS.
     EXEC SQL SELECT *
              INTO :EMP-NUMB, :EMP-FIRST-NAME,
                   :EMP-MID-INIT, :EMP-LAST-NAME,
                   :EMP-WORK-DEPT:WORK-DEPT-IND
              FROM VEMP
```

```
              WHERE EMPNO = :DEPT-MGR
     END-EXEC.
     IF SQLCODE = 100 THEN
         MOVE SPACES TO EMP-RECORD.
 2300-GETEMPREC-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* ADD AN EMPLOYEE                                               *
*---------------------------------------------------------------*
 3000-ADDEMP.
     CALL 'ISPLINK' USING I-VPUT, IDEN-VAR.
     PERFORM 3100-DISEMPDATA THRU 3100-DISEMPDATA-EXIT.
     CALL 'ISPLINK' USING I-VPUT, ADD-EMP-VARS.
     CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL.
     IF RETURN-CODE NOT EQUAL TO 8 THEN
                     EXEC SQL OPEN CURDEPTLOC END-EXEC
     PERFORM 3320-SETCURLOC THRU 3320-SETCURLOC-EXIT.
                     EXEC SQL CLOSE CURDEPTLOC END-EXEC
                     EXEC SQL OPEN DEPTLOC END-EXEC
     EXEC SQL FETCH DEPTLOC INTO :DEPT-LOC END-EXEC
                     EXEC SQL CLOSE DEPTLOC END-EXEC
        IF DEPT-LOC NOT EQUAL TO LOCATION THEN
             MOVE '216E' TO MSGCODE
             CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
             MOVE OUTMSG TO MSGS
        ELSE
             EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC
             MOVE SPACES TO SQLERRP
             EXEC SQL INSERT INTO VEMP
                      VALUES (:EMP-NUMB, :EMP-FIRST-NAME,
                              :EMP-MID-INIT, :EMP-LAST-NAME,
                              :EMP-WORK-DEPT)
             END-EXEC
             PERFORM 3200-ADDEMPCODES THRU 3200-ADDEMPCODES-EXIT
             EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR
             END-EXEC
             PERFORM 3300-GETDEPTREC THRU 3300-GETDEPTREC-EXIT
             CALL 'ISPLINK' USING I-VPUT, DEPT-VARS
             CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL.
 3000-ADDEMP-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* DISPLAY INPUT DATA ON PANEL                                   *
*---------------------------------------------------------------*
 3100-DISEMPDATA.
     MOVE SPACES TO DEPT-RECORD.
     MOVE SPACES TO EMP-RECORD.
     IF SEARCH-CRIT = 'EI' THEN
         MOVE DATAW TO EMP-NUMB
     ELSE
         IF SEARCH-CRIT = 'EN' THEN
             MOVE DATAW TO EMP-LAST-NAME.
 3100-DISEMPDATA-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* CHECK RETURN CODE FROM INSERT                                 *
*---------------------------------------------------------------*
 3200-ADDEMPCODES.
     IF SQLERRP = SPACES THEN
         MOVE '079E' TO MSGCODE
     ELSE
         IF SQLCODE = -803 THEN
             MOVE '005E' TO MSGCODE
         ELSE
             IF SQLCODE = -530 THEN
                 MOVE '200E' TO MSGCODE
             ELSE
                 IF SQLCODE = 0 THEN
                     MOVE '002I' TO MSGCODE
                 ELSE
                     GO TO L8000-P3-DBERROR.
     CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG.
     MOVE OUTMSG TO MSGS.
 3200-ADDEMPCODES-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* RETRIEVE DEPARTMENT INFO FOR NEW EMPLOYEE                     *
*---------------------------------------------------------------*
```

```
 3300-GETDEPTREC.
     CALL 'ISPLINK' USING I-VGET, ADD-EMP-VARS.
     EXEC SQL SELECT *
              INTO :DEPT-NUMB, :DEPT-NAME,
                   :DEPT-MGR:DEPT-MGR-IND,
                   :DEPT-ADMR, :DEPT-LOC
              FROM VHDEPT
              WHERE DEPTNO = :EMP-WORK-DEPT
     END-EXEC.
     IF SQLCODE = 100 THEN
         MOVE SPACES TO DEPT-RECORD
     ELSE
         PERFORM 3310-CHECKDEPTIND THRU 3310-CHECKDEPTIND-EXIT.
 3300-GETDEPTREC-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* IF MGRNO NULL, MOVE BLANKS INTO FIELD                         *
*---------------------------------------------------------------*
 3310-CHECKDEPTIND.
     IF DEPT-MGR-IND < 0 THEN
         MOVE SPACES TO DEPT-MGR.
 3310-CHECKDEPTIND-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* SET LOCATION TO CURRENT SERVER                                *
*---------------------------------------------------------------*
 3320-SETCURLOC.
     IF LOCATION EQUAL TO SPACES THEN
       EXEC SQL FETCH CURDEPTLOC
         INTO :LOCATION
       END-EXEC.
 3320-SETCURLOC-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* MOVE APPROPRIATE ACTION INTO ACTION-LIST                      *
*---------------------------------------------------------------*
 4000-ACTION.
     IF ACTION = 'E' THEN
         MOVE '  ERASE' TO ACTION-LIST
     ELSE
         IF ACTION = 'U' THEN
             MOVE ' UPDATE' TO ACTION-LIST
         ELSE
             MOVE 'DISPLAY' TO ACTION-LIST.
     MOVE 0 TO PERCENT-COUNTER.
     INSPECT DATAW
         TALLYING PERCENT-COUNTER FOR ALL '%'.
     IF PERCENT-COUNTER > 0 THEN
         INSPECT DATAW
             REPLACING ALL ' ' BY '%'.
 4000-ACTION-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* PERFORM ACTION ON DEPARTMENT OR DEPARTMENT STRUCTURE          *
*---------------------------------------------------------------*
 5000-DEPARTMENT.
     IF NOT (SEARCH-CRIT = 'DI' AND PERCENT-COUNTER = 0) THEN
         MOVE DATAW TO GENDATA
         PERFORM 5100-GENDEPT THRU 5100-GENDEPT-EXIT
             UNTIL GEND-EXIT = 'Y'
     ELSE
         IF OBJFLD = 'DE' THEN
             PERFORM 5200-DISPLAYDEPT THRU 5200-DISPLAYDEPT-EXIT
         ELSE
             PERFORM 5300-STRUCTURE THRU 5300-STRUCTURE-EXIT.
 5000-DEPARTMENT-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* GENERIC LIST OF DEPARTMENTS                                   *
*---------------------------------------------------------------*
 5100-GENDEPT.
     CALL 'ISPLINK' USING I-TBCREATE, DEPT-TABLE, W-BLANK,
         SEL-DEPT-VARS, I-NOWRITE, I-REPLACE.
     MOVE SPACE TO SEL-DEPT.
     PERFORM 5110-GETDEPTTAB THRU 5110-GETDEPTTAB-EXIT.
     CALL 'ISPLINK' USING I-TBQUERY, DEPT-TABLE, W-BLANK,
         W-BLANK, QROWS.
```

```
     IF NUMROWS = 1 AND GENDATA = DATAW THEN
         MOVE 'Y' TO SPECIAL-EXIT
         CALL 'ISPLINK' USING I-TBGET, DEPT-TABLE
         MOVE DEPT-NUMB TO DATAW
     ELSE
         MOVE 'N' TO SPECIAL-EXIT
         IF NUMROWS = 0 THEN
             PERFORM 5120-DEPTMSG THRU 5120-DEPTMSG-EXIT
             MOVE 'Y' TO GEND-EXIT
         ELSE
             CALL 'ISPLINK' USING I-VPUT, ACTL-VAR
             CALL 'ISPLINK' USING I-TBTOP, DEPT-TABLE
             CALL 'ISPLINK' USING I-TBDISPL, DEPT-TABLE,
                 GEND-PANEL
             IF RETURN-CODE = 8 THEN
                 MOVE 'Y' TO GEND-EXIT
             ELSE
                 IF ROWS-CHANGED > 0 THEN
                     CALL 'ISPLINK' USING I-TBGET, DEPT-TABLE
                     MOVE DEPT-NUMB TO DATAW
                 ELSE
                     MOVE 'Y' TO GEND-EXIT.
     IF GEND-EXIT = 'N' THEN
         IF OBJFLD = 'DE' THEN
             PERFORM 5200-DISPLAYDEPT THRU 5200-DISPLAYDEPT-EXIT
         ELSE
             PERFORM 5300-STRUCTURE THRU 5300-STRUCTURE-EXIT.
     IF SPECIAL-EXIT = 'Y' THEN
         MOVE 'Y' TO GEND-EXIT.
     CALL 'ISPLINK' USING I-TBCLOSE, DEPT-TABLE.
 5100-GENDEPT-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* CREATE TABLE OF DEPARTMENTS TO FIT SEARCH-CRIT                *
*---------------------------------------------------------------*
 5110-GETDEPTTAB.
     IF SEARCH-CRIT = 'DI' THEN
         EXEC SQL OPEN ALLDEPT1 END-EXEC
         MOVE SPACES TO SQLERRP
         PERFORM 5111-ALLDEPT1 THRU 5111-ALLDEPT1-EXIT
             UNTIL SQLCODE NOT EQUAL TO 0 OR GEND-EXIT = 'Y'
         EXEC SQL CLOSE ALLDEPT1 END-EXEC
     ELSE
         IF SEARCH-CRIT = 'DN' AND PERCENT-COUNTER > 0 THEN
             EXEC SQL OPEN ALLDEPT2 END-EXEC
             MOVE SPACES TO SQLERRP
             PERFORM 5112-ALLDEPT2 THRU 5112-ALLDEPT2-EXIT
                 UNTIL SQLCODE NOT EQUAL TO 0 OR GEND-EXIT = 'Y'
             EXEC SQL CLOSE ALLDEPT2 END-EXEC
         ELSE
             IF SEARCH-CRIT = 'DN' THEN
                 EXEC SQL OPEN ALLDEPT5 END-EXEC
                 MOVE SPACES TO SQLERRP
                 PERFORM 5113-ALLDEPT5 THRU 5113-ALLDEPT5-EXIT
                     UNTIL SQLCODE NOT EQUAL TO 0 OR
                           GEND-EXIT = 'Y'
                 EXEC SQL CLOSE ALLDEPT5 END-EXEC
             ELSE
                 IF SEARCH-CRIT = 'MI' AND
                     PERCENT-COUNTER > 0 THEN
                     EXEC SQL OPEN ALLDEPT3 END-EXEC
                     MOVE SPACES TO SQLERRP
                     PERFORM 5114-ALLDEPT3 THRU
                             5114-ALLDEPT3-EXIT
                             UNTIL SQLCODE NOT EQUAL TO 0 OR
                                   GEND-EXIT = 'Y'
                     EXEC SQL CLOSE ALLDEPT3 END-EXEC
                 ELSE
                     IF SEARCH-CRIT = 'MI' THEN
                         EXEC SQL OPEN ALLDEPT6 END-EXEC
                         MOVE SPACES TO SQLERRP
                         PERFORM 5115-ALLDEPT6 THRU
                                 5115-ALLDEPT6-EXIT
                                 UNTIL SQLCODE NOT EQUAL TO 0 OR
                                       GEND-EXIT = 'Y'
                         EXEC SQL CLOSE ALLDEPT6 END-EXEC
                     ELSE
                         IF SEARCH-CRIT = 'MN' AND
                             PERCENT-COUNTER > 0 THEN
                             EXEC SQL OPEN ALLDEPT4 END-EXEC
                             MOVE SPACES TO SQLERRP
```

```
                                   PERFORM 5116-ALLDEPT4 THRU
                                           5116-ALLDEPT4-EXIT
                                           UNTIL SQLCODE NOT EQUAL TO 0
                                               OR GEND-EXIT = 'Y'
                                   EXEC SQL CLOSE ALLDEPT4 END-EXEC
                               ELSE
                                   IF SEARCH-CRIT = 'MN' THEN
                                       EXEC SQL OPEN ALLDEPT7 END-EXEC
                                       MOVE SPACES TO SQLERRP
                                       PERFORM 5117-ALLDEPT7 THRU
                                               5117-ALLDEPT7-EXIT
                                               UNTIL SQLCODE NOT EQUAL
                                               TO 0 OR GEND-EXIT = 'Y'
                                       EXEC SQL CLOSE ALLDEPT7
                                       END-EXEC.
       5110-GETDEPTTAB-EXIT.
           EXIT.
      *
       5111-ALLDEPT1.
           EXEC SQL FETCH ALLDEPT1
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5111-ALLDEPT1-EXIT.
           EXIT.
      *
       5112-ALLDEPT2.
           EXEC SQL FETCH ALLDEPT2
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5112-ALLDEPT2-EXIT.
           EXIT.
      *
       5113-ALLDEPT5.
           EXEC SQL FETCH ALLDEPT5
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5113-ALLDEPT5-EXIT.
           EXIT.
      *
       5114-ALLDEPT3.
           EXEC SQL FETCH ALLDEPT3
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
```

```
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5114-ALLDEPT3-EXIT.
           EXIT.
      *
       5115-ALLDEPT6.
           EXEC SQL FETCH ALLDEPT6
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5115-ALLDEPT6-EXIT.
           EXIT.
      *
       5116-ALLDEPT4.
           EXEC SQL FETCH ALLDEPT4
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5116-ALLDEPT4-EXIT.
           EXIT.
      *
       5117-ALLDEPT7.
           EXEC SQL FETCH ALLDEPT7
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :MGR-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GEND-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 3310-CHECKDEPTIND THRU
                           3310-CHECKDEPTIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE.
       5117-ALLDEPT7-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * PRINT CORRECT 'DEPARTMENT NOT FOUND' MESSAGE                   *
      *----------------------------------------------------------------*
       5120-DEPTMSG.
           IF MSGCODE NOT EQUAL TO '079E' THEN
               IF ACTION = 'E' THEN
                   MOVE '016E' TO MSGCODE
               ELSE
                   IF ACTION = 'U' THEN
                       MOVE '017E' TO MSGCODE
                   ELSE
                       MOVE '011I' TO MSGCODE.
           CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
           MOVE OUTMSG TO MSGS.
       5120-DEPTMSG-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * DISPLAY A DEPARTMENT                                           *
      *----------------------------------------------------------------*
       5200-DISPLAYDEPT.
           MOVE SPACES TO DEPT-RECORD.
           MOVE SPACES TO EMP-RECORD.
           EXEC SQL OPEN DEPT1 END-EXEC.
           MOVE SPACES TO SQLERRP.
           EXEC SQL FETCH DEPT1 INTO :DEPT-NUMB, :DEPT-NAME,
                                     :DEPT-MGR:DEPT-MGR-IND,
```

```
                               :DEPT-ADMR, :DEPT-LOC,
                               :EMP-NUMB, :EMP-FIRST-NAME,
                               :EMP-MID-INIT, :EMP-LAST-NAME,
                               :EMP-WORK-DEPT:WORK-DEPT-IND
     END-EXEC.
     PERFORM 5210-DISDEPTACT THRU 5210-DISDEPTACT-EXIT.
 5200-DISPLAYDEPT-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* DISPLAY, ERASE, OR UPDATE DEPARTMENT                          *
*---------------------------------------------------------------*
 5210-DISDEPTACT.
     IF SQLERRP = SPACES THEN
         EXEC SQL CLOSE DEPT1 END-EXEC
         MOVE '079E' TO MSGCODE
         CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
         MOVE OUTMSG TO MSGS
     ELSE
         IF SQLCODE = 100 THEN
             EXEC SQL CLOSE DEPT1 END-EXEC
             PERFORM 5120-DEPTMSG THRU 5120-DEPTMSG-EXIT
         ELSE
             EXEC SQL CLOSE DEPT1 END-EXEC
             PERFORM 3310-CHECKDEPTIND THRU
                     3310-CHECKDEPTIND-EXIT
             CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL
             IF RETURN-CODE NOT EQUAL TO 8 THEN
                 IF ACTION = 'E' THEN
                     PERFORM 5220-ERASEDEPT THRU
                             5220-ERASEDEPT-EXIT
                 ELSE
                     IF ACTION = 'U' THEN
                         PERFORM 5230-UPDATEDEPT THRU
                                 5230-UPDATEDEPT-EXIT.
 5210-DISDEPTACT-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* ERASE A DEPARTMENT                                            *
*---------------------------------------------------------------*
 5220-ERASEDEPT.
     MOVE 1 TO DEPTPTR.
     MOVE 0 TO LISTPTR.
     MOVE DATAW TO DEPTS (DEPTPTR).
     PERFORM 5221-DELDEPTS THRU 5221-DELDEPTS-EXIT
         UNTIL DEPTPTR = 0.
     MOVE LISTPTR TO STACKTOP.
     PERFORM 5223-DELDEPEND THRU 5223-DELDEPEND-EXIT
         UNTIL LISTPTR = 0.
     EXEC SQL OPEN LOCS END-EXEC.
     MOVE 0 TO LOCPTR.
     PERFORM 2210-BUILDLOCTABLE THRU 2210-BUILDLOCTABLE-EXIT
         UNTIL SQLCODE NOT EQUAL TO 0.
     EXEC SQL CLOSE LOCS END-EXEC.
     MOVE LOCPTR TO LOCTOP.
     MOVE 0 TO LOCPTR.
     PERFORM 5224-DELETELOCS THRU 5224-DELETELOCS-EXIT
         UNTIL LOCPTR = LOCTOP.
     MOVE '013I' TO MSGCODE.
     CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG.
     MOVE OUTMSG TO MSGS.
     PERFORM 1100-CONNECT THRU 1100-CONNECT-EXIT.
 5220-ERASEDEPT-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
* ERASE DEPARTMENT FROM OTHER LOCATIONS                         *
*---------------------------------------------------------------*
 5221-DELDEPTS.
     ADD 1 TO LISTPTR.
     MOVE DEPTS (DEPTPTR) TO DEPTLIST (LISTPTR).
     MOVE DEPTS (DEPTPTR) TO CURDEPT.
     SUBTRACT 1 FROM DEPTPTR.
     EXEC SQL OPEN SUBDEPTS END-EXEC.
     PERFORM 5222-GETSUBDEPTS THRU 5222-GETSUBDEPTS-EXIT
         UNTIL SQLCODE NOT EQUAL TO 0.
     EXEC SQL CLOSE SUBDEPTS END-EXEC.
 5221-DELDEPTS-EXIT.
     EXIT.
*
*---------------------------------------------------------------*
```

```
      * BUILD TABLE OF DEPARTMENTS DEPENDENT ON ERASED DEPARTMENTS    *
      * AND DEPARTMENTS DEPENDENT ON THOSE DEPARTMENTS ETC.           *
      *---------------------------------------------------------------*
       5222-GETSUBDEPTS.
           EXEC SQL FETCH SUBDEPTS INTO :TEMPDEPT END-EXEC.
           IF SQLCODE = 0 THEN
               ADD 1 TO DEPTPTR
               MOVE TEMPDEPT TO DEPTS (DEPTPTR).
       5222-GETSUBDEPTS-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * ENOFORCE REFERENTIAL INTEGRITY RULE ON VHDEPT BY CASCADE      *
      * DELETING DEPARTMENTS DEPENDENT ON DELETED DEPARTMENTS         *
      *---------------------------------------------------------------*
       5223-DELDEPEND.
           MOVE DEPTLIST (LISTPTR) TO DELDEPT.
           EXEC SQL DELETE FROM VHDEPT
                    WHERE DEPTNO = :DELDEPT
           END-EXEC.
           SUBTRACT 1 FROM LISTPTR.
       5223-DELDEPEND-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * PERFORM CASCADE DELETE AT ALL LOCATIONS                       *
      *---------------------------------------------------------------*
       5224-DELETELOCS.
           IF LOCPTR < LOCTOP THEN
               ADD 1 TO LOCPTR
               MOVE LOCLIST (LOCPTR) TO TEMPLOC
               EXEC SQL CONNECT TO :TEMPLOC END-EXEC
               MOVE STACKTOP TO LISTPTR
               PERFORM 5223-DELDEPEND THRU 5223-DELDEPEND-EXIT
                   UNTIL LISTPTR = 0.
       5224-DELETELOCS-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * UPDATE A DEPARTMENT                                           *
      *---------------------------------------------------------------*
       5230-UPDATEDEPT.
           PERFORM 2300-GETEMPREC THRU 2300-GETEMPREC-EXIT.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           EXEC SQL UPDATE VHDEPT
                    SET DEPTNAME = :DEPT-NAME,
                        MGRNO    = :DEPT-MGR,
                        ADMRDEPT = :DEPT-ADMR,
                        LOCATION = :DEPT-LOC
                    WHERE DEPTNO = :DATAW
           END-EXEC.
           IF SQLCODE = -530 THEN
               UNSTRING SQLERRMC
                   DELIMITED BY HIGH-VALUE
                   INTO TOKEN
               IF TOKEN = 'RDD' THEN
                   MOVE '215E' TO MSGCODE
                   CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                   MOVE OUTMSG TO MSGS
               ELSE
                   IF TOKEN = 'RDE' THEN
                       MOVE '214E' TO MSGCODE
                       CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                       MOVE OUTMSG TO MSGS
                   ELSE
                       GO TO L8000-P3-DBERROR
               ELSE
                   IF SQLCODE NOT EQUAL TO 0 THEN
                       GO TO L8000-P3-DBERROR
                   ELSE
                       EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR
                       END-EXEC
                       EXEC SQL OPEN LOCS END-EXEC
                       MOVE 0 TO LOCPTR
                       PERFORM 2210-BUILDLOCTABLE THRU
                               2210-BUILDLOCTABLE-EXIT
                               UNTIL SQLCODE NOT EQUAL TO 0
                       EXEC SQL CLOSE LOCS END-EXEC
                       MOVE LOCPTR TO LOCTOP
                       MOVE 0 TO LOCPTR
                       PERFORM 5231-UPDATELOCS THRU
                               5231-UPDATELOCS-EXIT
```

```
                       UNTIL LOCPTR = LOCTOP
                   MOVE '014I' TO MSGCODE
                   CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                   MOVE OUTMSG TO MSGS
                   PERFORM 1100-CONNECT THRU 1100-CONNECT-EXIT
           EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR END-EXEC.
           CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL.
       5230-UPDATEDEPT-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * UPDATE DEPARTMENT TO VHDEPT VIEWS AT ALL LOCATIONS            *
      *---------------------------------------------------------------*
       5231-UPDATELOCS.
           IF LOCPTR < LOCTOP THEN
               ADD 1 TO LOCPTR
               MOVE LOCLIST (LOCPTR) TO TEMPLOC
               EXEC SQL CONNECT TO :TEMPLOC END-EXEC
               EXEC SQL UPDATE VHDEPT
                        SET DEPTNAME = :DEPT-NAME,
                            MGRNO    = :DEPT-MGR,
                            ADMRDEPT = :DEPT-ADMR,
                            LOCATION = :DEPT-LOC
                        WHERE DEPTNO = :DEPT-NUMB
               END-EXEC.
       5231-UPDATELOCS-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * DISPLAY DEPARTMENT STRUCTURE                                  *
      *---------------------------------------------------------------*
       5300-STRUCTURE.
           MOVE SPACES TO DEPT-RECORD.
           MOVE SPACES TO EMP-RECORD.
           MOVE SPACES TO DEPT1-RECORD.
           MOVE SPACES TO EMP1-RECORD.
           EXEC SQL OPEN DEPT1 END-EXEC.
           MOVE SPACES TO SQLERRP.
           EXEC SQL FETCH DEPT1 INTO :DEPT1-NUMB, :DEPT1-NAME,
                                     :DEPT1-MGR:DEPT1-MGR-IND,
                                     :DEPT1-ADMR, :DEPT1-LOC,
                                     :EMP-NUMB,
                                     :EMP1-FIRST-NAME,
                                     :EMP1-MID-INIT,
                                     :EMP1-LAST-NAME,
                                     :EMP1-WORK-DEPT:WORK1-DEPT-IND
           END-EXEC.
           PERFORM 5310-DISSTR THRU 5310-DISSTR-EXIT.
       5300-STRUCTURE-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * DISPLAY DEPARTMENTS REPORTING TO SELECTED DEPARTMENT          *
      *---------------------------------------------------------------*
       5310-DISSTR.
           IF SQLERRP = SPACES THEN
               EXEC SQL CLOSE DEPT1 END-EXEC
               MOVE '079E' TO MSGCODE
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
               MOVE OUTMSG TO MSGS
           ELSE
               IF SQLCODE = 100 THEN
                   EXEC SQL CLOSE DEPT1 END-EXEC
                   MOVE '011I' TO MSGCODE
                   CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                   MOVE OUTMSG TO MSGS
               ELSE
                   EXEC SQL CLOSE DEPT1 END-EXEC
                   PERFORM 5311-CHECKDEPT1IND THRU
                           5311-CHECKDEPT1IND-EXIT
                   CALL 'ISPLINK' USING I-TBCREATE, DS-TABLE, W-BLANK,
                       DS-VARS, I-NOWRITE, I-REPLACE
                   EXEC SQL OPEN DEPTSTR END-EXEC
                   PERFORM 5312-GETSTRTAB THRU 5312-GETSTRTAB-EXIT
                       UNTIL SQLCODE NOT EQUAL TO 0
                   EXEC SQL CLOSE DEPTSTR END-EXEC
                   CALL 'ISPLINK' USING I-TBTOP, DS-TABLE
                   CALL 'ISPLINK' USING I-TBDISPL, DS-TABLE, STR-PANEL
                   CALL 'ISPLINK' USING I-VPUT, HEAD-DEPT-VARS
                   CALL 'ISPLINK' USING I-TBCLOSE, DS-TABLE.
       5310-DISSTR-EXIT.
           EXIT.
```

```
      *
      *---------------------------------------------------------------*
      * IF MGRNO NULL, MOVE BLANKS INTO FIELD                         *
      *---------------------------------------------------------------*
       5311-CHECKDEPT1IND.
           IF DEPT1-MGR-IND < 0 THEN
               MOVE SPACES TO DEPT1-MGR.
       5311-CHECKDEPT1IND-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * CREATE LIST OF DEPARTMENTS REPORTING TO SELECTED DEPARTMENT   *
      *---------------------------------------------------------------*
       5312-GETSTRTAB.
           EXEC SQL FETCH DEPTSTR
                    INTO :DEPT-NUMB, :DEPT-NAME,
                         :DEPT-MGR:DEPT-MGR-IND,
                         :DEPT-ADMR, :DEPT-LOC,
                         :EMP-FIRST-NAME, :EMP-MID-INIT,
                         :EMP-LAST-NAME
           END-EXEC.
           IF SQLCODE = 0 THEN
               PERFORM 3310-CHECKDEPTIND THRU 3310-CHECKDEPTIND-EXIT
               CALL 'ISPLINK' USING I-TBADD, DS-TABLE.
       5312-GETSTRTAB-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * PERFORM ACTION ON EMPLOYEE                                    *
      *---------------------------------------------------------------*
       6000-EMPLOYEE.
           IF NOT (SEARCH-CRIT = 'EI' AND PERCENT-COUNTER = 0) THEN
               MOVE DATAW TO GENDATA
               PERFORM 6100-GENEMP THRU 6100-GENEMP-EXIT
                   UNTIL GENE-EXIT = 'Y'
           ELSE
               PERFORM 6200-DISPLAYEMP THRU 6200-DISPLAYEMP-EXIT.
       6000-EMPLOYEE-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * GENERIC LIST OF EMPLOYEES                                     *
      *---------------------------------------------------------------*
       6100-GENEMP.
           CALL 'ISPLINK' USING I-TBCREATE, EMP-TABLE, W-BLANK,
               SEL-EMP-VARS, I-NOWRITE, I-REPLACE.
           MOVE SPACE TO SEL-EMP.
           PERFORM 6110-GETEMPTAB THRU 6110-GETEMPTAB-EXIT.
           CALL 'ISPLINK' USING I-TBQUERY, EMP-TABLE, W-BLANK, W-BLANK,
               QROWS.
           IF NUMROWS = 1 AND DATAW = GENDATA THEN
               MOVE 'Y' TO SPECIAL-EXIT
               CALL 'ISPLINK' USING I-TBGET, EMP-TABLE
               MOVE EMP-NUMB TO DATAW
           ELSE
               MOVE 'N' TO SPECIAL-EXIT
               IF NUMROWS = 0 THEN
                   PERFORM 6120-EMPMSG THRU 6120-EMPMSG-EXIT
                   MOVE 'Y' TO GENE-EXIT
               ELSE
                   CALL 'ISPLINK' USING I-VPUT, ACTL-VAR
                   CALL 'ISPLINK' USING I-TBTOP, EMP-TABLE
                   CALL 'ISPLINK' USING I-TBDISPL, EMP-TABLE,
                       GENE-PANEL
                   IF RETURN-CODE = 8 THEN
                       MOVE 'Y' TO GENE-EXIT
                   ELSE
                       IF ROWS-CHANGED > 0 THEN
                           CALL 'ISPLINK' USING I-TBGET, EMP-TABLE
                           MOVE EMP-NUMB TO DATAW
                       ELSE
                           MOVE 'Y' TO GENE-EXIT.
           IF GENE-EXIT = 'N' THEN
               PERFORM 6200-DISPLAYEMP THRU 6200-DISPLAYEMP-EXIT.
           IF SPECIAL-EXIT = 'Y' THEN
               MOVE 'Y' TO GENE-EXIT.
           CALL 'ISPLINK' USING I-TBCLOSE, EMP-TABLE.
       6100-GENEMP-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * CREATE TABLE OF EMPLOYEES TO FIT SEARCH-CRIT                  *
```

```
      *----------------------------------------------------------------*
       6110-GETEMPTAB.
           IF SEARCH-CRIT = 'EI' THEN
               EXEC SQL OPEN ALLEMP1 END-EXEC
               MOVE SPACES TO SQLERRP
               PERFORM 6111-ALLEMP1 THRU 6111-ALLEMP1-EXIT
                   UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y'
               EXEC SQL CLOSE ALLEMP1 END-EXEC
           ELSE
               IF SEARCH-CRIT = 'EN' AND PERCENT-COUNTER > 0 THEN
                   EXEC SQL OPEN ALLEMP2 END-EXEC
                   MOVE SPACES TO SQLERRP
                   PERFORM 6112-ALLEMP2 THRU 6112-ALLEMP2-EXIT
                       UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y'
                   EXEC SQL CLOSE ALLEMP2 END-EXEC
               ELSE
                   EXEC SQL OPEN ALLEMP3 END-EXEC
                   MOVE SPACES TO SQLERRP
                   PERFORM 6113-ALLEMP3 THRU 6113-ALLEMP3-EXIT
                       UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y'
                   EXEC SQL CLOSE ALLEMP3 END-EXEC.
       6110-GETEMPTAB-EXIT.
           EXIT.
      *
       6111-ALLEMP1.
           EXEC SQL FETCH ALLEMP1
                    INTO :EMP-NUMB, :EMP-NAME,
                         :EMP-WORK-DEPT:WORK-DEPT-IND,
                         :DEPT-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GENE-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 6114-CHECKEMPIND THRU 6114-CHECKEMPIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, EMP-TABLE.
       6111-ALLEMP1-EXIT.
           EXIT.
      *
       6112-ALLEMP2.
           EXEC SQL FETCH ALLEMP2
                    INTO :EMP-NUMB, :EMP-NAME,
                         :EMP-WORK-DEPT:WORK-DEPT-IND,
                         :DEPT-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GENE-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 6114-CHECKEMPIND THRU 6114-CHECKEMPIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, EMP-TABLE.
       6112-ALLEMP2-EXIT.
           EXIT.
      *
       6113-ALLEMP3.
           EXEC SQL FETCH ALLEMP3
                    INTO :EMP-NUMB, :EMP-NAME,
                         :EMP-WORK-DEPT:WORK-DEPT-IND,
                         :DEPT-NAME
           END-EXEC.
           IF SQLERRP = SPACES THEN
               MOVE '079E' TO MSGCODE
               MOVE 'Y' TO GENE-EXIT
           ELSE
               IF SQLCODE = 0 THEN
                   PERFORM 6114-CHECKEMPIND THRU 6114-CHECKEMPIND-EXIT
                   CALL 'ISPLINK' USING I-TBADD, EMP-TABLE.
       6113-ALLEMP3-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * IF WORKDEPT NULL, MOVE BLANKS INTO FIELD                       *
      *----------------------------------------------------------------*
       6114-CHECKEMPIND.
           IF WORK-DEPT-IND < 0 THEN
               MOVE SPACES TO EMP-WORK-DEPT.
       6114-CHECKEMPIND-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
```

```
      * PRINT CORRECT 'EMPLOYEE NOT FOUND' MESSAGE                    * 00201600
      *---------------------------------------------------------------* 00201700
       6120-EMPMSG.                                                     00201800
           IF MSGCODE NOT EQUAL TO '079E' THEN                          00201900
               IF ACTION = 'E' THEN                                     00202000
                   MOVE '006E' TO MSGCODE                               00202100
               ELSE                                                     00202200
                   IF ACTION = 'U' THEN                                 00202300
                       MOVE '007E' TO MSGCODE                           00202400
                   ELSE                                                 00202500
                       MOVE '001I' TO MSGCODE.                          00202600
           CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG                 00202700
           MOVE OUTMSG TO MSGS.                                         00202800
       6120-EMPMSG-EXIT.                                                00202900
           EXIT.                                                        00203000
      *                                                                 00203100
      *---------------------------------------------------------------* 00203200
      * DISPLAY AN EMPLOYEE                                           * 00203300
      *---------------------------------------------------------------* 00203400
       6200-DISPLAYEMP.                                                 00203500
           MOVE SPACES TO DEPT-RECORD.                                  00203600
           MOVE SPACES TO EMP-RECORD.                                   00203700
           EXEC SQL OPEN EMP1 END-EXEC.                                 00203800
           MOVE SPACES TO SQLERRP.                                      00203900
           EXEC SQL FETCH EMP1 INTO :DEPT-NUMB, :DEPT-NAME,             00204000
                                    :DEPT-MGR:DEPT-MGR-IND,             00204100
                                    :DEPT-ADMR, :DEPT-LOC,              00204200
                                    :EMP-NUMB, :EMP-FIRST-NAME,         00204300
                                    :EMP-MID-INIT, :EMP-LAST-NAME,      00204400
                                    :EMP-WORK-DEPT:WORK-DEPT-IND        00204500
           END-EXEC.                                                    00204600
           PERFORM 6210-DISEMPACT THRU 6210-DISEMPACT-EXIT.             00204700
       6200-DISPLAYEMP-EXIT.                                            00204800
           EXIT.                                                        00204900
      *                                                                 00205000
      *---------------------------------------------------------------* 00205100
      * DISPLAY, ERASE, OR UPDATE EMPLOYEE                            * 00205200
      *---------------------------------------------------------------* 00205300
       6210-DISEMPACT.                                                  00205400
           IF SQLERRP = SPACES THEN                                     00205500
               EXEC SQL CLOSE EMP1 END-EXEC                             00205600
               MOVE '079E' TO MSGCODE                                   00205700
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG             00205800
               MOVE OUTMSG TO MSGS                                      00205900
           ELSE                                                         00206000
               IF SQLCODE = 100 THEN                                    00206100
                   EXEC SQL CLOSE EMP1 END-EXEC                         00206200
                   PERFORM 6120-EMPMSG THRU 6120-EMPMSG-EXIT            00206300
               ELSE                                                     00206400
                   EXEC SQL CLOSE EMP1 END-EXEC                         00206500
                   PERFORM 3310-CHECKDEPTIND THRU                       00206600
                           3310-CHECKDEPTIND-EXIT                       00206700
                   CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL            00206800
                   IF RETURN-CODE NOT EQUAL TO 8 THEN                   00206900
                       IF ACTION = 'E' THEN                             00207000
                           PERFORM 6220-ERASEEMP THRU                   00207100
                                   6220-ERASEEMP-EXIT                   00207200
                       ELSE                                             00207300
                           IF ACTION = 'U' THEN                         00207400
                               PERFORM 6230-UPDATEEMP THRU              00207500
                                       6230-UPDATEEMP-EXIT.             00207600
       6210-DISEMPACT-EXIT.                                             00207700
           EXIT.                                                        00207800
      *                                                                 00207900
      *---------------------------------------------------------------* 00208000
      * ERASE AN EMPLOYEE                                             * 00208100
      *---------------------------------------------------------------* 00208200
       6220-ERASEEMP.                                                   00208300
           EXEC SQL DELETE FROM VEMP                                    00208400
                    WHERE EMPNO = :DATAW                                00208500
           END-EXEC.                                                    00208600
           IF SQLCODE = 0 THEN                                          00208700
               MOVE '003I' TO MSGCODE                                   00208800
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG             00208900
               MOVE OUTMSG TO MSGS.                                     00209000
       6220-ERASEEMP-EXIT.                                              00209100
           EXIT.                                                        00209200
      *                                                                 00209300
      *---------------------------------------------------------------* 00209400
      * UPDATE AN EMPLOYEE                                            * 00209500
      *---------------------------------------------------------------* 00209600
       6230-UPDATEEMP.                                                  00209700
```

```
     PERFORM 3300-GETDEPTREC THRU 3300-GETDEPTREC-EXIT.
                     EXEC SQL OPEN CURDEPTLOC END-EXEC
     PERFORM 3320-SETCURLOC THRU 3320-SETCURLOC-EXIT.
                     EXEC SQL CLOSE CURDEPTLOC END-EXEC
     IF DEPT-LOC NOT EQUAL TO LOCATION THEN
         MOVE '217E' TO MSGCODE
         CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
         MOVE OUTMSG TO MSGS
     ELSE
         EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC
         EXEC SQL UPDATE VEMP
                  SET FIRSTNME = :EMP-FIRST-NAME,
                      MIDINIT  = :EMP-MID-INIT,
                      LASTNAME = :EMP-LAST-NAME,
                      WORKDEPT = :EMP-WORK-DEPT
                  WHERE EMPNO = :DATAW
         END-EXEC
         IF SQLCODE = -530 THEN
             MOVE '203E' TO MSGCODE
             CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
             MOVE OUTMSG TO MSGS
         ELSE
             IF SQLCODE = 0 THEN
                 MOVE '004I' TO MSGCODE
                 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                 MOVE OUTMSG TO MSGS
             ELSE
                 GO TO L8000-P3-DBERROR.
     CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL.
 6230-UPDATEEMP-EXIT.
     EXIT.
*
*--------------------------------------------------------------*
* DB2 ERROR PROCESSING                                         *
*--------------------------------------------------------------*
 L8000-P3-DBERROR.

        MOVE SQLCAID TO SQLCAID-VALUE.
        MOVE SQLCABC TO CONV.
        MOVE CONV TO SQLCABC-VALUE.
        MOVE SQLCODE TO CONV.
        MOVE CONV TO SQLCODE-VALUE, SQLCODE-MSG.
        MOVE SQLERRML TO CONV.
        MOVE CONV TO SQLERRML-VALUE.
        MOVE SQLERRMC TO SQLERRMC-VALUE.
        MOVE SQLERRP TO SQLERRP-VALUE.
        MOVE SQLERRD (1) TO CONV.
        MOVE CONV TO SQLERRD1-VALUE.
        MOVE SQLERRD (2) TO CONV.
        MOVE CONV TO SQLERRD2-VALUE.
        MOVE SQLERRD (3) TO CONV.
        MOVE CONV TO SQLERRD3-VALUE.
        MOVE SQLERRD (4) TO CONV.
        MOVE CONV TO SQLERRD4-VALUE.
        MOVE SQLERRD (5) TO CONV.
        MOVE CONV TO SQLERRD5-VALUE.
        MOVE SQLERRD (6) TO CONV.
        MOVE CONV TO SQLERRD6-VALUE.
        MOVE SQLWARN0 TO SQLWARN0-VALUE.
        MOVE SQLWARN1 TO SQLWARN1-VALUE.
        MOVE SQLWARN2 TO SQLWARN2-VALUE.
        MOVE SQLWARN3 TO SQLWARN3-VALUE.
        MOVE SQLWARN4 TO SQLWARN4-VALUE.
        MOVE SQLWARN5 TO SQLWARN5-VALUE.
        MOVE SQLWARN6 TO SQLWARN6-VALUE.
        MOVE SQLWARN7 TO SQLWARN7-VALUE.
        MOVE SQLWARN8 TO SQLWARN8-VALUE.
        MOVE SQLWARN9 TO SQLWARN9-VALUE.
        MOVE SQLWARNA TO SQLWARNA-VALUE.
        MOVE SQLSTATE TO SQLSTATE-VALUE.

        OPEN OUTPUT MSGOUT.
        WRITE MSGREC FROM SQLCA-LINE0.
        WRITE MSGREC FROM SQLCA-LINE1.
        WRITE MSGREC FROM SQLCA-LINE2.
        WRITE MSGREC FROM SQLCA-LINE3.
        WRITE MSGREC FROM SQLCA-LINE4.
        WRITE MSGREC FROM SQLCA-LINE5.
        WRITE MSGREC FROM SQLCA-LINE6.
        WRITE MSGREC FROM SQLCA-LINE7.
        WRITE MSGREC FROM SQLCA-LINE8.
        WRITE MSGREC FROM SQLCA-LINE9.
```

```
           WRITE MSGREC FROM SQLCA-LINE10.                              00218000
           WRITE MSGREC FROM SQLCA-LINE11.                              00218100
           WRITE MSGREC FROM SQLCA-LINE12.                              00218200
           WRITE MSGREC FROM SQLCA-LINE13.                              00218300
           WRITE MSGREC FROM SQLCA-LINE14.                              00218400
           CLOSE MSGOUT.                                                00218500
                                                                        00218600
        GOBACK.                                                         00218700
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8SC3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
        IDENTIFICATION DIVISION.                                        00000100
       *-----------------------                                         00000200
        PROGRAM-ID. DSN8SC3.                                            00000300
                                                                        00000400
       *---------------------------------------------------------------* 00000500
       *                                                               * 00000600
       *   MODULE NAME = DSN8SC3                                       * 00000700
       *                                                               * 00000800
       *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                  * 00000900
       *                      PHONE APPLICATION                        * 00001000
       *                      ISPF                                     * 00001100
       *                      COBOL                                    * 00001200
       *                                                               * 00001300
       *COPYRIGHT = 5615-DB2 (C) COPYRIGHT 1982, 2013 IBM CORP.        * 00001400
       *SEE COPYRIGHT INSTRUCTIONS                                     * 00001500
       *LICENSED MATERIALS - PROPERTY OF IBM                           * 00001600
       *                                                               * 00001700
       *STATUS = STATUS = VERSION 11                                   * 00001800
       *                                                               * 00001900
       *   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND     * 00002000
       *              UPDATES THEM IF DESIRED.                         * 00002100
       *                                                               * 00002200
       *   NOTES =                                                     * 00002300
       *      DEPENDENCIES = TWO ISPF PANELS ARE REQUIRED:             * 00002400
       *                     DSN8SSL AND DSN8SSN                       * 00002500
       *      RESTRICTIONS = NONE                                      * 00002600
       *                                                               * 00002700
       *   MODULE TYPE = VS COBOL II PROGRAM                           * 00002800
       *      PROCESSOR   = DB2 PRECOMPILER, VS COBOL II               * 00002900
       *      MODULE SIZE = SEE LINKEDIT                               * 00003000
       *      ATTRIBUTES  = NOT REENTRANT OR REUSABLE                  * 00003100
       *                                                               * 00003200
       *   ENTRY POINT =  DSN8SC3                                      * 00003300
       *      PURPOSE = SEE FUNCTION                                   * 00003400
       *      LINKAGE = INVOKED FROM ISPF                              * 00003500
       *                                                               * 00003600
       *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:   * 00003700
       *              INPUT-MESSAGE:                                   * 00003800
       *                                                               * 00003900
       *                     SYMBOLIC LABEL/NAME = DSN8SSL             * 00004000
       *                     DESCRIPTION = PHONE MENU 1 (SELECT)       * 00004100
       *                                                               * 00004200
       *                     SYMBOLIC LABEL/NAME = DSN8SSN             * 00004300
       *                     DESCRIPTION = PHONE MENU 2 (LIST)         * 00004400
       *                                                               * 00004500
       *                     SYMBOLIC LABEL/NAME = VPHONE              * 00004600
       *                     DESCRIPTION = VIEW OF TELEPHONE DATA      * 00004700
       *                                                               * 00004800
       *                     SYMBOLIC LABEL/NAME = VEMPLP              * 00004900
       *                     DESCRIPTION = VIEW OF EMPLOYEE DATA       * 00005000
       *                                                               * 00005100
       *      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                 * 00005200
       *               OUTPUT-MESSAGE:                                 * 00005300
       *                                                               * 00005400
       *                     SYMBOLIC LABEL/NAME = DSN8SSL             * 00005500
       *                     DESCRIPTION = PHONE MENU 1 (SELECT)       * 00005600
       *                                                               * 00005700
       *                     SYMBOLIC LABEL/NAME = DSN8SSN             * 00005800
       *                     DESCRIPTION = PHONE MENU 2 (LIST)         * 00005900
       *                                                               * 00006000
       *   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION               * 00006100
```

```
*                                                               * 00006200
*   EXIT-ERROR =                                                * 00006300
*                                                               * 00006400
*      RETURN CODE = NONE                                       * 00006500
*                                                               * 00006600
*      ABEND CODES =  NONE                                      * 00006700
*                                                               * 00006800
*                                                               * 00006900
*      ERROR-MESSAGES =                                         * 00007000
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED           * 00007100
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE              * 00007200
*            DSN8060E - SQL ERROR, RETURN CODE IS:              * 00007300
*            DSN8079E - CONNECTION TO DB2 NOT ESTABLISHED       * 00007400
*                                                               * 00007500
*   EXTERNAL REFERENCES =                                       * 00007600
*      ROUTINES/SERVICES =                                      * 00007700
*         DSN8MCG             - ERROR MESSAGE ROUTINE           * 00007800
*         ISPLINK             - ISPF SERVICES ROUTINE           * 00007900
*                                                               * 00008000
*      DATA-AREAS =                                             * 00008100
*         NONE                                                  * 00008200
*                                                               * 00008300
*      CONTROL-BLOCKS =                                         * 00008400
*         SQLCA               - SQL COMMUNICATION AREA          * 00008500
*                                                               * 00008600
*   TABLES = NONE                                               * 00008700
*                                                               * 00008800
*                                                               * 00008900
*   CHANGE-ACTIVITY:                                            * 00009000
*                                                               * 00009100
*   CHECK SQLERRP FOR NON-BLANKS TO ENSURE CONNECTION     V2R3  * 00009200
*       HAS BEEN ESTABLISHED.  ISSUE 079E IF NOT.               * 00009300
*                                                               * 00009400
*  *PSEUDOCODE*                                                 * 00009500
*                                                               * 00009600
*   SET UP RETURN CODE HANDLING               0000-PROGRAM-START* 00009700
*   DO UNTIL NO MORE TERMINAL INPUT                             * 00009800
*      GET PANEL INPUT                        1000-MAIN-LOOP    * 00009900
*      DETERMINE PROCESSING REQUEST           2000-GET-TYPE     * 00010000
*        -IF "LIST ALL" (*):                  3000-LIST-ALL     * 00010100
*            FETCH FIRST RECORD                                 * 00010200
*            CREATE ISPF TABLE                                  * 00010300
*            DO UNTIL NO MORE RECORDS:                          * 00010400
*               STORE RECORD IN TABLE         3500-LIST-AND-GET * 00010500
*               GET ANOTHER RECORD                              * 00010600
*        -IF "LIST GENERIC" (%):              4000-LIST-GENERIC * 00010700
*            FETCH FIRST RECORD                                 * 00010800
*            CREATE ISPF TABLE                                  * 00010900
*            DO UNTIL NO MORE MATCHING RECORDS:                 * 00011000
*               STORE RECORD IN TABLE         4500-LIST-AND-GET * 00011100
*               GET ANOTHER RECORD                              * 00011200
*        -IF "LIST SPECIFIC:                  5000-LIST-SPECIFIC* 00011300
*            FETCH FIRST RECORD                                 * 00011400
*            CREATE ISPF TABLE                                  * 00011500
*            DO UNTIL NO MORE MATCHING RECORDS:                 * 00011600
*               STORE RECORD IN TABLE         5500-LIST-AND-GET * 00011700
*               GET ANOTHER RECORD                              * 00011800
*      DISPLAY PHONE LIST ON SCREEN           6000-DISPLAY-LIST * 00011900
*      IF UPDATE REQUESTED                    6500-UPDATE-LOOP  * 00012000
*         UPDATE PHONE RECORDS                7000-UPDATE       * 00012100
*---------------------------------------------------------------* 00012200
 ENVIRONMENT DIVISION.                                            00012300
 DATA DIVISION.                                                   00012400
 WORKING-STORAGE SECTION.                                         00012500
*---------------------------------------------------------------* 00012600
 77  COIBM                       PIC X(54) VALUE IS               00012700
     'COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987'.    00012800
 77  SEL-EXIT                    PIC X(01).                       00012900
 77  DIS-EXIT                    PIC X(01).                       00013000
 77  DISPLAY-TABLE               PIC X(01).                       00013100
 77  MORE-CHANGES                PIC X(01).                       00013200
 77  ROWS-CHANGED                PIC 9(04).                       00013300
 77  PERCENT-COUNTER             PIC S9(4)  COMP.                 00013400
 77  MODULE                      PIC X(07)  VALUE 'DSN8SC3'.      00013500
 77  MSGCODE                     PIC X(04).                       00013600
 77  W-BLANK                     PIC X(01)  VALUE ' '.            00013700
 77  MSGS-VAR                    PIC X(08)  VALUE 'DSN8MSGS'.     00013800
 77  FI-VAR                      PIC X(08)  VALUE 'FNAMEI  '.     00013900
 77  LI-VAR                      PIC X(08)  VALUE 'LNAMEI  '.     00014000
*--------------------------------------------------------------*  00014100
* ISPF DIALOG VARIABLE NAMES                                   *  00014200
*--------------------------------------------------------------*  00014300
```

```
     EXEC SQL INCLUDE SQLCA END-EXEC.                             00014400
 01  LNAMEW                      PIC X(15).                       00014500
 01  FNAMEW                      PIC X(12).                       00014600
 01  LIST-PANEL-VARIABLES.                                        00014700
     03  CH-VAR                  PIC X(08)  VALUE 'ZTDSELS '.     00014800
     03  FN-VAR                  PIC X(08)  VALUE 'FNAMED  '.     00014900
     03  MI-VAR                  PIC X(08)  VALUE 'MINITD  '.     00015000
     03  LN-VAR                  PIC X(08)  VALUE 'LNAMED  '.     00015100
     03  PN-VAR                  PIC X(08)  VALUE 'PNOD    '.     00015200
     03  EN-VAR                  PIC X(08)  VALUE 'ENOD    '.     00015300
     03  WD-VAR                  PIC X(08)  VALUE 'WDEPTD  '.     00015400
     03  WN-VAR                  PIC X(08)  VALUE 'WNAMED  '.     00015500
     03  TABLE-NAME              PIC X(08)  VALUE 'DSN8TABL'.     00015600
     03  SEL-VARS                PIC X(20)  VALUE IS              00015700
     '( FNAMEI LNAMEI )   '.                                      00015800
     03  DIS-VARS                PIC X(56)  VALUE IS              00015900
     '( ZTDSELS FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED )'.  00016000
     03  EMP-VARS                PIC X(48)  VALUE IS              00016100
     '( FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED )'.          00016200
 01  PANEL-VARIABLE-LENGTHS.                                      00016300
     03  CH-VAR-STG              PIC 9(06)  COMP VALUE 04.        00016400
     03  FN-VAR-STG              PIC 9(06)  COMP VALUE 12.        00016500
     03  MI-VAR-STG              PIC 9(06)  COMP VALUE 01.        00016600
     03  LN-VAR-STG              PIC 9(06)  COMP VALUE 15.        00016700
     03  PN-VAR-STG              PIC 9(06)  COMP VALUE 04.        00016800
     03  EN-VAR-STG              PIC 9(06)  COMP VALUE 06.        00016900
     03  WD-VAR-STG              PIC 9(06)  COMP VALUE 03.        00017000
     03  WN-VAR-STG              PIC 9(06)  COMP VALUE 36.        00017100
     03  FI-VAR-STG              PIC 9(06)  COMP VALUE 12.        00017200
     03  LI-VAR-STG              PIC 9(06)  COMP VALUE 15.        00017300
     03  MSGS-VAR-STG            PIC 9(06)  COMP VALUE 79.        00017400
*---------------------------------------------------------------* 00017500
* ISPF DIALOG SERVICES DECLARATIONS                             * 00017600
*---------------------------------------------------------------* 00017700
 01  I-VDEFINE               PIC X(08)  VALUE 'VDEFINE '.         00017800
 01  I-VGET                  PIC X(08)  VALUE 'VGET    '.         00017900
 01  I-VPUT                  PIC X(08)  VALUE 'VPUT    '.         00018000
 01  I-DISPLAY               PIC X(08)  VALUE 'DISPLAY '.         00018100
 01  I-TBDISPL               PIC X(08)  VALUE 'TBDISPL '.         00018200
 01  I-TBTOP                 PIC X(08)  VALUE 'TBTOP   '.         00018300
 01  I-TBCREATE              PIC X(08)  VALUE 'TBCREATE'.         00018400
 01  I-TBCLOSE               PIC X(08)  VALUE 'TBCLOSE '.         00018500
 01  I-TBADD                 PIC X(08)  VALUE 'TBADD   '.         00018600
 01  I-TBPUT                 PIC X(08)  VALUE 'TBPUT   '.         00018700
*---------------------------------------------------------------* 00018800
* ISPF CALL MODIFIERS                                           * 00018900
*---------------------------------------------------------------* 00019000
 01  I-NOWRITE               PIC X(08)  VALUE 'NOWRITE '.         00019100
 01  I-REPLACE               PIC X(08)  VALUE 'REPLACE '.         00019200
 01  I-CHAR                  PIC X(08)  VALUE 'CHAR    '.         00019300
*---------------------------------------------------------------* 00019400
* ISPF PANEL NAMES                                              * 00019500
*---------------------------------------------------------------* 00019600
 01  SEL-PANEL               PIC X(08)  VALUE 'DSN8SSL '.         00019700
 01  DIS-PANEL               PIC X(08)  VALUE 'DSN8SSN '.         00019800
*---------------------------------------------------------------* 00019900
* LOCAL-VARIABLES                                               * 00020000
*---------------------------------------------------------------* 00020100
 01  LOCAL-VARIABLES.                                             00020200
     03  LNAMEI                  PIC X(15) VALUE SPACES.          00020300
     03  FNAMEI                  PIC X(12) VALUE SPACES.          00020400
     03  CONVSQL                 PIC S9(15) COMP-3.               00020500
     03  OUTMSG                  PIC X(69).                       00020600
     03  TMSG REDEFINES OUTMSG.                                   00020700
         05  TMSGTXT             PIC X(46).                       00020800
         05  FILLER              PIC X(23).                       00020900
     03  MSGS                    PIC X(79) VALUE SPACES.          00021000
     03  MSGS-DETAIL REDEFINES MSGS.                              00021100
         05  OUT-MESSAGE         PIC X(46).                       00021200
         05  SQL-CODE            PIC +(04).                       00021300
         05  FILLER              PIC X(29).                       00021400
*---------------------------------------------------------------* 00021500
* EMPLOYEE RECORD - IO AREA                                     * 00021600
*---------------------------------------------------------------* 00021700
 01  EMP-RECORD.                                                  00021800
     02  EMPLAST                 PIC X(15).                       00021900
     02  EMP-FIRST-NAME          PIC X(12).                       00022000
     02  EMP-MIDDLE-INITIAL      PIC X(01).                       00022100
     02  EMPPHONE                PIC X(04).                       00022200
     02  EMPNUMB                 PIC X(06).                       00022300
     02  EMP-DEPT-NUMBER         PIC X(03).                       00022400
     02  EMP-DEPTNAME            PIC X(36).                       00022500
```

```
*---------------------------------------------------------------*
* SQL DECLARATION FOR VIEW PHONE                                *
*---------------------------------------------------------------*
     EXEC SQL DECLARE VPHONE TABLE
         (LASTNAME      VARCHAR(15)          ,
          FIRSTNAME     VARCHAR(12)          ,
          MIDDLEINITIAL     CHAR(1)          ,
          PHONENUMBER       CHAR(4)          ,
          EMPLOYEENUMBER    CHAR(6)          ,
          DEPTNUMBER        CHAR(3)  NOT NULL,
          DEPTNAME       VARCHAR(36) NOT NULL) END-EXEC.
*---------------------------------------------------------------*
* STRUCTURE FOR PHONE RECORD                                    *
*---------------------------------------------------------------*
 01  PPHONE.
     02 LAST-NAME                PIC X(15).
     02 FIRST-NAME               PIC X(12).
     02 MIDDLE-INITIAL           PIC X(01).
     02 PHONE-NUMBER             PIC X(04).
     02 EMPLOYEE-NUMBER          PIC X(06).
     02 DEPT-NUMBER              PIC X(03).
     02 DEPTNAME                 PIC X(36).
*---------------------------------------------------------------*
* SQL DECLARATION FOR VIEW VEMPLP                               *
*---------------------------------------------------------------*
     EXEC SQL DECLARE VEMPLP TABLE
          (EMPLOYEENUMBER    CHAR(6)          ,
           PHONENUMBER       CHAR(4)) END-EXEC.
*---------------------------------------------------------------*
* SQL CURSORS                                                   *
*---------------------------------------------------------------*
     EXEC SQL DECLARE TELE1 CURSOR FOR
         SELECT *
         FROM VPHONE
     END-EXEC.
*
     EXEC SQL DECLARE TELE2 CURSOR FOR
         SELECT *
         FROM  VPHONE
         WHERE LASTNAME  LIKE :LNAMEW
           AND FIRSTNAME LIKE :FNAMEW
     END-EXEC.
*
     EXEC SQL DECLARE TELE3 CURSOR FOR
             SELECT *
             FROM  VPHONE
             WHERE LASTNAME  =    :LNAMEW
               AND FIRSTNAME LIKE :FNAMEW
     END-EXEC.
*
     EJECT
 PROCEDURE DIVISION.
*---------------------------------------------------------------*
* SQL RETURN CODE HANDLING                                      *
*---------------------------------------------------------------*
     EXEC SQL WHENEVER SQLERROR   GOTO L8000-P3-DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO L8000-P3-DBERROR END-EXEC.
     EXEC SQL WHENEVER NOT FOUND  CONTINUE END-EXEC.
*
*---------------------------------------------------------------*
* DEFINE COBOL - SPF VARIABLES                                  *
*---------------------------------------------------------------*
 0000-PROGRAM-START.
     CALL 'ISPLINK' USING I-VDEFINE, CH-VAR, ROWS-CHANGED,
                 I-CHAR, CH-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, FN-VAR, EMP-FIRST-NAME,
                 I-CHAR, FN-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, MI-VAR, EMP-MIDDLE-INITIAL,
                 I-CHAR, MI-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, LN-VAR, EMPLAST,
                 I-CHAR, LN-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, PN-VAR, EMPPHONE,
                 I-CHAR, PN-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, EN-VAR, EMPNUMB,
                 I-CHAR, EN-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, WD-VAR, EMP-DEPT-NUMBER,
                 I-CHAR, WD-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, WN-VAR, EMP-DEPTNAME,
                 I-CHAR, WN-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, FI-VAR, FNAMEI,
                 I-CHAR, FI-VAR-STG.
     CALL 'ISPLINK' USING I-VDEFINE, LI-VAR, LNAMEI,
```

```
                        I-CHAR, LI-VAR-STG.
           CALL 'ISPLINK' USING I-VDEFINE, MSGS-VAR, MSGS,
                        I-CHAR, MSGS-VAR-STG.
      *
      *---------------------------------------------------------------*
      * MAIN PROGRAM                                                  *
      *---------------------------------------------------------------*
           MOVE 'N' TO SEL-EXIT.
           PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT
                UNTIL SEL-EXIT = 'Y'.
           MOVE 0 TO RETURN-CODE.
           GOBACK.
      *
       1000-MAIN-LOOP.
           CALL 'ISPLINK' USING I-DISPLAY, SEL-PANEL.
           MOVE SPACES TO MSGS.
           MOVE SPACES TO OUTMSG.
           IF RETURN-CODE = 8
               MOVE 'Y' TO SEL-EXIT
           ELSE
               MOVE 'N' TO DISPLAY-TABLE
               CALL 'ISPLINK' USING I-VGET, SEL-VARS
               MOVE LNAMEI TO LNAMEW
               MOVE FNAMEI TO FNAMEW
               PERFORM 2000-GET-TYPE THRU 2000-GET-TYPE-EXIT
               IF DISPLAY-TABLE = 'Y'
                   PERFORM 6000-DISPLAY-LIST
                   THRU 6000-DISPLAY-LIST-EXIT.
           CALL 'ISPLINK' USING I-VPUT MSGS-VAR.
       1000-MAIN-LOOP-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * DETERMINE PROCESSING REQUEST                                  *
      *---------------------------------------------------------------*
       2000-GET-TYPE.
           IF LNAMEW = '*'
               PERFORM 3000-LIST-ALL
                  THRU 3000-LIST-ALL-EXIT
           ELSE
               UNSTRING LNAMEW
                   DELIMITED BY SPACE
                   INTO      LNAMEW
               UNSTRING FNAMEW
                   DELIMITED BY SPACE
                   INTO      FNAMEW
               INSPECT FNAMEW
                   REPLACING ALL ' ' BY '%'
               MOVE 0 TO PERCENT-COUNTER
               INSPECT LNAMEW
                   TALLYING PERCENT-COUNTER FOR ALL '%'
               IF PERCENT-COUNTER > 0
                   INSPECT LNAMEW
                       REPLACING ALL ' ' BY '%'
                   PERFORM 4000-LIST-GENERIC
                      THRU 4000-LIST-GENERIC-EXIT
               ELSE
                   PERFORM 5000-LIST-SPECIFIC
                      THRU 5000-LIST-SPECIFIC-EXIT.
       2000-GET-TYPE-EXIT.
           EXIT.
      *
      *---------------------------------------------------------------*
      * LIST ALL EMPLOYEES                                            *
      *---------------------------------------------------------------*
       3000-LIST-ALL.
           EXEC SQL OPEN TELE1 END-EXEC.
           MOVE SPACES TO SQLERRP.
           EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
           IF SQLERRP = SPACES
               MOVE '079E' TO MSGCODE
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
               MOVE OUTMSG TO MSGS
           ELSE
               IF SQLCODE = 100
                   MOVE '008I' TO MSGCODE
                   CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                   MOVE OUTMSG TO MSGS
               ELSE
                   MOVE 'Y' TO DISPLAY-TABLE
                   CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME,
                       W-BLANK, EMP-VARS, I-NOWRITE, I-REPLACE
```

```
             PERFORM  3500-LIST-AND-GET
                 THRU 3500-LIST-AND-GET-EXIT
                 UNTIL SQLCODE NOT EQUAL 0.
         EXEC SQL CLOSE TELE1 END-EXEC.
     3000-LIST-ALL-EXIT.
         EXIT.
    *
     3500-LIST-AND-GET.
         MOVE PPHONE TO EMP-RECORD.
         CALL 'ISPLINK' USING I-TBADD, TABLE-NAME.
         EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
     3500-LIST-AND-GET-EXIT.
         EXIT.
    *
    *----------------------------------------------------------------*
    * GENERIC LIST OF EMPLOYEES                                      *
    *----------------------------------------------------------------*
     4000-LIST-GENERIC.
         EXEC SQL OPEN TELE2 END-EXEC.
         MOVE SPACES TO SQLERRP.
         EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.
         IF SQLERRP = SPACES
             MOVE '079E' TO MSGCODE
             CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
             MOVE OUTMSG TO MSGS
         ELSE
             IF SQLCODE = 100
                 MOVE '008I' TO MSGCODE
                 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                 MOVE OUTMSG TO MSGS
             ELSE
                 MOVE 'Y' TO DISPLAY-TABLE
                 CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME, W-BLANK,
                     EMP-VARS, I-NOWRITE, I-REPLACE
                 PERFORM  4500-LIST-AND-GET
                     THRU 4500-LIST-AND-GET-EXIT
                     UNTIL SQLCODE NOT EQUAL 0.
         EXEC SQL CLOSE TELE2 END-EXEC.
     4000-LIST-GENERIC-EXIT.
         EXIT.
    *
     4500-LIST-AND-GET.
         MOVE PPHONE TO EMP-RECORD.
         CALL 'ISPLINK' USING I-TBADD, TABLE-NAME.
         EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.
     4500-LIST-AND-GET-EXIT.
         EXIT.
    *----------------------------------------------------------------*
    * SPECIFIC LIST OF EMPLOYEES                                     *
    *----------------------------------------------------------------*
     5000-LIST-SPECIFIC.
         EXEC SQL OPEN TELE3 END-EXEC.
         MOVE SPACES TO SQLERRP.
         EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.
         IF SQLERRP = SPACES
             MOVE '079E' TO MSGCODE
             CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
             MOVE OUTMSG TO MSGS
         ELSE
             IF SQLCODE = 100
                 MOVE '008I' TO MSGCODE
                 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
                 MOVE OUTMSG TO MSGS
             ELSE
                 MOVE 'Y' TO DISPLAY-TABLE
                 CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME,
                     W-BLANK, EMP-VARS, I-NOWRITE, I-REPLACE
                 PERFORM  5500-LIST-AND-GET
                     THRU 5500-LIST-AND-GET-EXIT
                     UNTIL SQLCODE NOT EQUAL 0.
         EXEC SQL CLOSE TELE3 END-EXEC.
     5000-LIST-SPECIFIC-EXIT.
         EXIT.
    *
     5500-LIST-AND-GET.
         MOVE PPHONE TO EMP-RECORD.
         CALL 'ISPLINK' USING I-TBADD, TABLE-NAME.
         EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.
     5500-LIST-AND-GET-EXIT.
         EXIT.
    *
    *----------------------------------------------------------------*
```

```
      * DISPLAY EMPLOYEE PHONE NUMBERS                                 *
      *----------------------------------------------------------------*
       6000-DISPLAY-LIST.
           EXEC SQL WHENEVER SQLERROR   CONTINUE END-EXEC.
           EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
           CALL 'ISPLINK' USING I-TBTOP, TABLE-NAME.
           CALL 'ISPLINK' USING I-TBDISPL, TABLE-NAME, DIS-PANEL.
           IF RETURN-CODE NOT EQUAL 8
               CALL 'ISPLINK' USING I-VGET, DIS-VARS
               PERFORM 6500-UPDATE-LOOP THRU 6500-UPDATE-LOOP-EXIT.
       6000-DISPLAY-LIST-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * DETERMINE IF UPDATE HAS BEEN REQUESTED                         *
      *----------------------------------------------------------------*
       6500-UPDATE-LOOP.
           IF ROWS-CHANGED > 0
               MOVE 'Y' TO MORE-CHANGES
               PERFORM 7000-UPDATE THRU 7000-UPDATE-EXIT
                   UNTIL MORE-CHANGES = 'N'.
           CALL 'ISPLINK' USING I-TBCLOSE, TABLE-NAME.
       6500-UPDATE-LOOP-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * UPDATE EMPLOYEE PHONE NUMBERS                                  *
      *----------------------------------------------------------------*
       7000-UPDATE.
           EXEC SQL UPDATE VEMPLP
               SET PHONENUMBER = :EMPPHONE
               WHERE EMPLOYEENUMBER = :EMPNUMB END-EXEC.
           IF SQLCODE NOT EQUAL 0
               MOVE '060E' TO MSGCODE
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
               MOVE OUTMSG TO TMSG
               MOVE TMSGTXT TO OUT-MESSAGE
               MOVE SQLCODE TO CONVSQL
               MOVE CONVSQL TO SQL-CODE
               EXEC SQL ROLLBACK END-EXEC
               MOVE 'N' TO MORE-CHANGES
           ELSE
               MOVE '004I' TO MSGCODE
               CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG
               MOVE OUTMSG TO MSGS
               CALL 'ISPLINK' USING I-TBPUT, TABLE-NAME
               IF ROWS-CHANGED > 1
                   CALL 'ISPLINK' USING I-TBDISPL, TABLE-NAME
                   CALL 'ISPLINK' USING I-VGET, DIS-VARS
               ELSE MOVE 'N' TO MORE-CHANGES.
       7000-UPDATE-EXIT.
           EXIT.
      *
      *----------------------------------------------------------------*
      * DB2 ERROR PROCESSING                                           *
      *----------------------------------------------------------------*
       L8000-P3-DBERROR.
           MOVE '060E' TO MSGCODE.
           CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG.
           MOVE OUTMSG TO TMSG.
           MOVE TMSGTXT TO OUT-MESSAGE.
           MOVE SQLCODE TO CONVSQL.
           MOVE CONVSQL TO SQL-CODE.
           CALL 'ISPLINK' USING I-VPUT, MSGS-VAR.
           GOBACK.
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8SP3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
DSN8SP3: PROC OPTIONS (MAIN);
/*********************************************************************
*                                                                    *
*   MODULE NAME = DSN8SP3                                            *
```

```
*                                                                     *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                        *
*                      PHONE APPLICATION                              *
*                      ISPF                                           *
*                      PL/I                                           *
*                                                                     *
*     COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1991          *
*     SEE COPYRIGHT INSTRUCTIONS                                      *
*     LICENSED MATERIALS - PROPERTY OF IBM                            *
*                                                                     *
*     STATUS = VERSION 2 RELEASE 3, LEVEL 0                           *
*                                                                     *
*   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND           *
*              UPDATES THEM IF DESIRED.                               *
*                                                                     *
*   NOTES =                                                           *
*      DEPENDENCIES = TWO ISPF PANELS ARE REQUIRED:                   *
*                     DSN8SSL AND DSN8SSN                             *
*      RESTRICTIONS = NONE                                            *
*                                                                     *
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                             *
*      PROCESSOR   = DB2 PRECOMPILER, PL/I OPTIMIZER                  *
*      MODULE SIZE = SEE LINKEDIT                                     *
*      ATTRIBUTES  = REENTRANT                                        *
*                                                                     *
*   ENTRY POINT =  DSN8SP3                                            *
*      PURPOSE = SEE FUNCTION                                         *
*      LINKAGE = INVOKED FROM ISPF                                    *
*                                                                     *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:         *
*              INPUT-MESSAGE:                                         *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = DSN8SSL                   *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)             *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = DSN8SSN                   *
*                     DESCRIPTION = PHONE MENU 2 (LIST)               *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = VPHONE                    *
*                     DESCRIPTION = VIEW OF TELEPHONE INFORMATION     *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = VEMPLP                    *
*                     DESCRIPTION = VIEW OF EMPLOYEE INFORMATION      *
*                                                                     *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                       *
*               OUTPUT-MESSAGE:                                       *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = DSN8SSL                   *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)             *
*                                                                     *
*                     SYMBOLIC LABEL/NAME = DSN8SSN                   *
*                     DESCRIPTION = PHONE MENU 2 (LIST)               *
*                                                                     *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION                     *
*                                                                     *
*   EXIT-ERROR =                                                      *
*                                                                     *
*      RETURN CODE = NONE                                             *
*                                                                     *
*      ABEND CODES =  NONE                                            *
*                                                                     *
*                                                                     *
*      ERROR-MESSAGES =                                               *
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED                 *
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE                    *
*            DSN8060E - SQL ERROR, RETURN CODE IS:                    *
*            DSN8079E - CONNECTION TO DB2 NOT ESTABLISHED             *
*                                                                     *
*   EXTERNAL REFERENCES =                                             *
*      ROUTINES/SERVICES =                                            *
*         DSN8MPG             - ERROR MESSAGE ROUTINE                 *
*         ISPLINK             - ISPF SERVICES ROUTINE                 *
*                                                                     *
*      DATA-AREAS =                                                   *
*         NONE                                                        *
*                                                                     *
*      CONTROL-BLOCKS =                                               *
*         SQLCA               - SQL COMMUNICATION AREA                *
*                                                                     *
*   TABLES = NONE                                                     *
*                                                                     *
*                                                                     *
```

```
*   CHANGE-ACTIVITY:                                                   *
*                                                                      *
*   CHECK SQLERRP FOR NON-BLANKS TO ENSURE CONNECTION          V2R3  *
*       HAS BEEN ESTABLISHED.  ISSUE 079E IF NOT.                      *
*                                                                      *
*  *PSEUDOCODE*                                                        *
*                                                                      *
*      PROCEDURE                                                       *
*         DO WHILE NOT EXIT-PRESSED                                    *
*            CALL GET-TYPE                                             *
*            CALL GET-LIST                                             *
*            CALL DISPLAY-LIST                                         *
*                                                                      *
*         GET_TYPE:                                                    *
*            IF LASTNAME IS '*'                                        *
*               TYPE = 'ALL'                                           *
*            ELSE                                                      *
*               IF LASTNAME CONTAINS '%'                               *
*                  TYPE = 'GENERIC'                                    *
*               ELSE                                                   *
*                  TYPE = 'SPECIFIC'                                   *
*                                                                      *
*         GET_LIST:                                                    *
*            CASE (TYPE)                                               *
*               SUBCASE ('ALL')                                        *
*                 GET ALL EMPLOYEES                                    *
*               SUBCASE ('GENERIC')                                    *
*                  GET GENERIC EMPLOYEES                               *
*               SUBCASE ('GENERIC')                                    *
*                  GET SPECIFIC EMPLOYEES                              *
*            ENDSUB                                                    *
*                                                                      *
*         DISPLAY_LIST:                                                *
*            DISPLAY LIST                                              *
*            IF NOT EXIT-PRESSED                                       *
*               UPDATE PHONE NUMBER(S)                                 *
*               WRITE CONFIRMATION MESSAGE                             *
*                                                                      *
*         P3_DBERROR:                                                  *
*            IF SQL ERROR OCCURS THEN                                  *
*              FORMAT ERROR MESSAGE                                    *
*              ROLLBACK                                                *
*            END                                                       *
* END.                                                                 *
**********************************************************************/

/********************************************************************/
/* DECLARATION FOR BUILTIN FUNCTIONS                                */
/********************************************************************/

DCL ADDR      BUILTIN;
DCL INDEX     BUILTIN;
DCL PLIRETC   BUILTIN;
DCL PLIRETV   BUILTIN;
DCL STG       BUILTIN;
DCL SUBSTR    BUILTIN;
DCL TRANSLATE BUILTIN;

/********************************************************************/
/* MESSAGE ROUTINE DECLARATIONS                                     */
/********************************************************************/

DCL  DSN8MPG  EXTERNAL ENTRY;

DCL  MODULE       CHAR (7)  INIT('DSN8SP3'); /* EXECUTING PROGRAM   */
DCL  OUTMSG       CHAR (69);                 /* MESSAGE TEXT        */

1/********************************************************************/
/* ISPF DIALOG VARIABLE NAMES                                       */
/********************************************************************/

/* SELECTION AND LIST PANEL VARIABLES                               */
DCL MSGS_VAR     CHAR(08) STATIC INIT('DSN8MSGS'); /*PANEL MSG FIELD*/

/* SELECTION PANEL VARIABLES                                        */
DCL FI_VAR       CHAR(08) STATIC INIT('FNAMEI  '); /*FIRST NAME VAR */
DCL LI_VAR       CHAR(08) STATIC INIT('LNAMEI  '); /*LAST NAME VAR  */

/* LIST PANEL VARIABLES                                             */
DCL CH_VAR       CHAR(08) STATIC INIT('ZTDSELS '); /*# ROWS CHANGED */
DCL FN_VAR       CHAR(08) STATIC INIT('FNAMED  '); /*FIRST NAME VAR */
DCL MI_VAR       CHAR(08) STATIC INIT('MINITD  '); /*MID INIT VAR   */
```

```
 DCL LN_VAR       CHAR(08) STATIC INIT('LNAMED  '); /*LAST NAME VAR  */
 DCL PN_VAR       CHAR(08) STATIC INIT('PNOD    '); /*PHONE NUM VAR  */
 DCL EN_VAR       CHAR(08) STATIC INIT('ENOD    '); /*EMPL NUM VAR   */
 DCL WD_VAR       CHAR(08) STATIC INIT('WDEPTD  '); /*WORK DEPT VAR  */
 DCL WN_VAR       CHAR(08) STATIC INIT('WNAMED  '); /*DEPT NAME VAR  */
 DCL TABLE_NAME   CHAR(08) STATIC INIT('DSN8TABL'); /*TABLE NAME VAR */
 DCL SEL_VARS     CHAR(20) STATIC                   /*SELECTION VARS */
     INIT('( FNAMEI LNAMEI )   ');
 DCL DIS_VARS     CHAR(56) STATIC                   /*DISPLAY VARS   */
     INIT('( ZTDSELS FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED )');
 DCL EMP_VARS     CHAR(48) STATIC                   /*DISPLAY VARS   */
     INIT('( FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED )');

 /*******************************************************************/
 /* ISPF DIALOG SERVICES DECLARATIONS                               */
 /*******************************************************************/
 /* PROGRAM NAME                                                    */
 DCL ISPLINK     EXTERNAL ENTRY OPTIONS(ASM INTER RETCODE);

 /* ISPF DIALOG SERVICE TYPES                                       */
 DCL I_VDEFINE   CHAR(8) STATIC INIT('VDEFINE ');
 DCL I_VGET      CHAR(8) STATIC INIT('VGET    ');
 DCL I_VPUT      CHAR(8) STATIC INIT('VPUT    ');
 DCL I_DISPLAY   CHAR(8) STATIC INIT('DISPLAY ');
 DCL I_TBDISPL   CHAR(8) STATIC INIT('TBDISPL ');
 DCL I_TBTOP     CHAR(8) STATIC INIT('TBTOP   ');
 DCL I_TBCREATE  CHAR(8) STATIC INIT('TBCREATE');
 DCL I_TBCLOSE   CHAR(8) STATIC INIT('TBCLOSE ');
 DCL I_TBADD     CHAR(8) STATIC INIT('TBADD   ');
 DCL I_TBPUT     CHAR(8) STATIC INIT('TBPUT   ');

 /* ISPF CALL MODIFIERS                                             */
 DCL I_NOWRITE   CHAR(8) STATIC INIT('NOWRITE');
 DCL I_REPLACE   CHAR(8) STATIC INIT('REPLACE');
 DCL I_CHAR      CHAR(8) STATIC INIT('CHAR');

 /* PANEL NAMES                                                     */
 DCL SEL_PANEL   CHAR(8) STATIC INIT('DSN8SSL'); /* SELECTION PANEL  */
 DCL DIS_PANEL   CHAR(8) STATIC INIT('DSN8SSN'); /* LIST PANEL       */

 /* LOCAL VARIABLES FOR ISPF VARIABLES                              */
 DCL LNAMEI      CHAR(15) INIT(' '); /* LAST-NAME INPUT             */
 DCL FNAMEI      CHAR(12) INIT(' '); /* FIRST-NAME INPUT            */

 DCL MSGS        CHAR(79) INIT(' '); /* MESSAGE FOR ISPF PANEL      */

 DCL 1 EMP_RECORD,                   /* PANEL DISPLAY INFORMATION    */
       2 LASTNAME         CHAR (15),
       2 FIRSTNAME        CHAR (12),
       2 MIDDLEINITIAL    CHAR (1),
       2 PHONENUMBER      CHAR (4),
       2 EMPLOYEENUMBER   CHAR (6),
       2 DEPTNUMBER       CHAR (3),
       2 DEPTNAME         CHAR (36);

1/*******************************************************************/
 /* DECLARATION FOR PROGRAM LOGIC                                   */
 /*******************************************************************/

 /* CONSTANTS                                                       */
 DCL YES          BIT(1) STATIC INIT('1'B);
 DCL NO           BIT(1) STATIC INIT('0'B);
 DCL ZERO         FIXED BIN(31,0) STATIC INIT(0);

 /* FLAGS                                                           */
 DCL SEL_EXIT     BIT(1);            /* EXIT PRESSED? FLAG          */
 DCL DIS_EXIT     BIT(1);            /* EXIT PRESSED? FLAG          */
 DCL DIS_TABLE    BIT(1);            /* DISPLAY-TABLE? FLAG         */
 DCL MORE_CHANGES BIT(1);            /* MORE CHANGES TO PROCESS?    */

 /* DATA VARIABLES                                                  */
 DCL ROWS_CHANGED PIC'9999';
 DCL TYPE         CHAR(8);           /* TYPE OF LIST                */
 DCL LNAMES       CHAR(15);          /* LAST NAME SELECTION VALUE   */
 DCL FNAMES       CHAR(12);          /* FIRST NAME SELECTION VALUE  */

1/*******************************************************************/
 /* SQL DECLARATIONS                                                */
 /*******************************************************************/

 /*                                  SQL COMMUNICATION AREA         */
0EXEC SQL INCLUDE SQLCA;
```

```
DCL SQL_PIC        PIC'-999';

/*                                  SQL DECLARATION FOR VIEW PHONE  */
EXEC SQL DECLARE VPHONE TABLE
       (LASTNAME       VARCHAR(15),
        FIRSTNAME      VARCHAR(12),
        MIDDLEINITIAL  CHAR(1),
        PHONENUMBER    CHAR(4),
        EMPLOYEENUMBER CHAR(6),
        DEPTNUMBER     CHAR(3)  NOT NULL,
        DEPTNAME       VARCHAR(36) NOT NULL);
                                /* STUCTURE FOR PHONE RECORD       */
DCL 1 PPHONE,
      2 LASTNAME         CHAR (15) VAR,
      2 FIRSTNAME        CHAR (12) VAR,
      2 MIDDLEINITIAL    CHAR (1),
      2 PHONENUMBER      CHAR (4),
      2 EMPLOYEENUMBER   CHAR (6),
      2 DEPTNUMBER       CHAR (3),
      2 DEPTNAME         CHAR (36) VAR;
                                /* SQL DECLARATION FOR VIEW VEMPLP*/
EXEC SQL DECLARE VEMPLP TABLE
       (EMPLOYEENUMBER    CHAR(6),
        PHONENUMBER       CHAR(4));

/*******************************************************************/
/* CURSOR DECLARATIONS                                             */
/*******************************************************************/

EXEC SQL DECLARE TELE1 CURSOR FOR
         SELECT *
         FROM VPHONE;

EXEC SQL DECLARE TELE2 CURSOR FOR
         SELECT *
         FROM  VPHONE
         WHERE LASTNAME LIKE :LNAMES
           AND FIRSTNAME LIKE :FNAMES;

EXEC SQL DECLARE TELE3 CURSOR FOR
         SELECT *
         FROM  VPHONE
         WHERE LASTNAME  =    :LNAMES
           AND FIRSTNAME LIKE :FNAMES;

1/*******************************************************************/
/* SQL RETURN CODE HANDLING                                        */
/*******************************************************************/

EXEC SQL WHENEVER SQLERROR   GOTO P3_DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;
EXEC SQL WHENEVER NOT FOUND  CONTINUE;

/*******************************************************************/
/* DEFINE PL/I - ISPF VARIABLES                                    */
/*******************************************************************/
CALL ISPLINK(I_VDEFINE, CH_VAR, ROWS_CHANGED,
             I_CHAR, STG(ROWS_CHANGED));
CALL ISPLINK(I_VDEFINE, FN_VAR, EMP_RECORD.FIRSTNAME,
             I_CHAR, STG(EMP_RECORD.FIRSTNAME));
CALL ISPLINK(I_VDEFINE, MI_VAR, EMP_RECORD.MIDDLEINITIAL,
             I_CHAR, STG(EMP_RECORD.MIDDLEINITIAL));
CALL ISPLINK(I_VDEFINE, LN_VAR, EMP_RECORD.LASTNAME,
             I_CHAR, STG(EMP_RECORD.LASTNAME));
CALL ISPLINK(I_VDEFINE, PN_VAR, EMP_RECORD.PHONENUMBER,
             I_CHAR, STG(EMP_RECORD.PHONENUMBER));
CALL ISPLINK(I_VDEFINE, EN_VAR, EMP_RECORD.EMPLOYEENUMBER,
             I_CHAR, STG(EMP_RECORD.EMPLOYEENUMBER));
CALL ISPLINK(I_VDEFINE, WD_VAR, EMP_RECORD.DEPTNUMBER,
             I_CHAR, STG(EMP_RECORD.DEPTNUMBER));
CALL ISPLINK(I_VDEFINE, WN_VAR, EMP_RECORD.DEPTNAME,
             I_CHAR, STG(EMP_RECORD.DEPTNAME));
CALL ISPLINK(I_VDEFINE, FI_VAR, FNAMEI,
             I_CHAR, STG(FNAMEI));
CALL ISPLINK(I_VDEFINE, LI_VAR, LNAMEI,
             I_CHAR, STG(LNAMEI));
CALL ISPLINK(I_VDEFINE, MSGS_VAR, MSGS,
             I_CHAR, STG(MSGS));

1/*******************************************************************/
/* MAIN PROGRAM                                                    */
/*******************************************************************/
```

```
 SEL_EXIT = '0'B;                       /* INITIALIZE EXIT BIT       */

 DO WHILE (^SEL_EXIT);                  /* DO WHILE NOT EXIT         */
   CALL ISPLINK(I_DISPLAY, SEL_PANEL);
   MSGS = ' ';                          /* RESET THE MSG FIELD       */
   OUTMSG = ' ';                        /* RESET THE MSG FIELD       */
   SEL_EXIT = (PLIRETV = 8);            /* SEL_EXIT = TRUE IF RC=8   */

   /*****************************************************************/
   /* EXIT WAS NOT SPECIFIED SO PROCESS THE REQUEST                 */
   /*****************************************************************/

   IF ^SEL_EXIT THEN                    /* IF USER PRESSED ENTER     */
     DO;
       DIS_TABLE = NO;                  /* INIT FLAG TO NO           */
       CALL ISPLINK(I_VGET, SEL_VARS);
       LNAMES = LNAMEI;                 /* COPY INPUT TO WORKING VAR */
       FNAMES = FNAMEI;                 /* COPY INPUT TO WORKING VAR */
       CALL GET_TYPE;                   /* DETERMINE LIST TYPE       */
       CALL GET_LIST;                   /* GET LIST OF EMPLOYEES     */
       IF DIS_TABLE THEN
           CALL DISPLAY_LIST;
     END;                               /* END IF USER PRESSED ENTER */

   CALL ISPLINK(I_VPUT, MSGS_VAR);      /* SET PANEL MESSAGE         */
 END;                                   /* END DO WHILE NOT EXIT     */

 CALL PLIRETC(ZERO);                    /* SET EXIT RETURN CODE TO 0 */
 RETURN;

1/*******************************************************************/
 /* GET TYPE OF LIST                                                */
 /*******************************************************************/

 GET_TYPE: PROCEDURE;

   IF LNAMES = '*' THEN                          /* LIST DIRECTORY   */
       TYPE = 'ALL';
   ELSE
     IF INDEX(LNAMES, '%') > 0 THEN
       DO;                                       /* GENERIC LIST     */
         TYPE = 'GENERIC';
         LNAMES = TRANSLATE(LNAMES, '%', ' ');   /* CHG SPACES TO %  */
         FNAMES = TRANSLATE(FNAMES, '%', ' ');   /* CHG SPACES TO %  */
       END;                                      /* END IF GENERIC   */
     ELSE
       DO;                                       /* SPECIFIC NAME    */
         TYPE = 'SPECIFIC';
         FNAMES = TRANSLATE(FNAMES, '%', ' ');   /* CHG SPACES TO %  */
       END;                                      /* END IF SPECIFIC  */

   END GET_TYPE;

1/*******************************************************************/
 /* GET LIST OF EMPLOYEES                                           */
 /*******************************************************************/

 GET_LIST: PROCEDURE;

   SQLERRP = ' ';                 /* CONNECTION CHECK: INIT SQLERRP  */

   SELECT (TYPE);                 /* OPEN CURSOR & GET FIRST RECORD  */
     WHEN ('ALL')                 /* FOR ALL EMPLOYEES               */
       DO;
         EXEC SQL OPEN TELE1;
         EXEC SQL FETCH TELE1
                  INTO  :PPHONE;
       END;
     WHEN ('GENERIC')             /* FOR GENERIC EMPLOYEES           */
       DO;
         EXEC SQL OPEN TELE2;
         EXEC SQL FETCH TELE2
                  INTO  :PPHONE;
       END;
     OTHERWISE                    /* FOR SPECIFIC EMPLOYEE(S)        */
       DO;
         EXEC SQL OPEN TELE3;
         EXEC SQL FETCH TELE3
                  INTO  :PPHONE;
       END;
     END;                         /* SELECT                          */
```

```
 /*******************************************************************/
 /* NO EMPLOYEE FULFILLED THE REQUEST                               */
 /*******************************************************************/

   SELECT;
     WHEN (SQLERRP = ' ')                  /* NO CONNECTION TO DB2   */
       DO;
         CALL DSN8MPG (MODULE, '079E', OUTMSG);
         MSGS = OUTMSG;                    /* SET ISPF ERROR MESSAGE */
       END;                                /* END NO EMPLOYEE FOUND  */
     WHEN (SQLCODE = 100)
       DO;
         CALL DSN8MPG (MODULE, '008I', OUTMSG);
         MSGS = OUTMSG;                    /* SET ISPF ERROR MESSAGE */
       END;                                /* END NO EMPLOYEE FOUND  */
     OTHERWISE

 /*******************************************************************/
 /* EMPLOYEES EXIST THAT FULFILL THE REQUEST.  DISPLAY THEM.        */
 /*******************************************************************/

     DO;                                   /* BUILD RESULT TABLE     */
       DIS_TABLE = YES;
       CALL ISPLINK(I_TBCREATE, TABLE_NAME, ' ', EMP_VARS, I_NOWRITE,
                    I_REPLACE);
       DO WHILE (SQLCODE = 0);               /* WHILE MORE ENTRIES   */
         EMP_RECORD = PPHONE, BY NAME;
         CALL ISPLINK(I_TBADD, TABLE_NAME);  /* ADD TO ISPF TABLE    */
         SELECT (TYPE);                      /* GET NEXT RECORD      */
           WHEN ('ALL')
              EXEC SQL FETCH TELE1 INTO :PPHONE;
           WHEN ('GENERIC')
              EXEC SQL FETCH TELE2 INTO :PPHONE;
           OTHERWISE
              EXEC SQL FETCH TELE3 INTO :PPHONE;
         END;                                /* END SELECT           */
       END;                                  /* END WHILE MORE       */
     END;                                    /* END EMPLOYEE FOUND   */
   END;                                      /* END SELECT           */
 /*******************************************************************/
 /* CLOSE THE CURSORS                                               */
 /*******************************************************************/

   SELECT (TYPE);
     WHEN ('ALL')
        EXEC SQL CLOSE TELE1;
     WHEN ('GENERIC')
        EXEC SQL CLOSE TELE2;
     OTHERWISE
        EXEC SQL CLOSE TELE3;
     END;

   END GET_LIST;

1/*******************************************************************/
 /* DISPLAY/UPDATE EMPLOYEE PHONE NUMBERS                           */
 /*******************************************************************/

 DISPLAY_LIST: PROCEDURE;

   EXEC SQL WHENEVER SQLERROR   CONTINUE;  /* CHANGE ERROR HANDLING  */
   EXEC SQL WHENEVER SQLWARNING CONTINUE;  /* FOR UPDATE             */

   CALL ISPLINK(I_TBTOP, TABLE_NAME);
   CALL ISPLINK(I_TBDISPL, TABLE_NAME, DIS_PANEL);

   DIS_EXIT = (PLIRETV = 8);               /* WAS EXIT PRESSED?      */
   IF ^DIS_EXIT THEN                       /* IF EXIT NOT PRESSED    */
     DO;
       CALL ISPLINK(I_VGET, DIS_VARS);
       MORE_CHANGES = (ROWS_CHANGED > 0);  /* ANY CHANGES?           */
       DO WHILE(MORE_CHANGES);             /* FIND PHONE NUM UPDATES */
         EXEC SQL UPDATE VEMPLP            /* PERFORM UPDATE         */
                  SET PHONENUMBER = :EMP_RECORD.PHONENUMBER
                  WHERE EMPLOYEENUMBER = :EMP_RECORD.EMPLOYEENUMBER;
         IF SQLCODE ^= 0 THEN              /* IF UPDATE FAILED       */
           DO;
             CALL DSN8MPG(MODULE, '060E', OUTMSG);
             SQL_PIC = SQLCODE;
             MSGS = SUBSTR(OUTMSG,1,46) || SQL_PIC;
             EXEC SQL ROLLBACK;
```

```
            MORE_CHANGES = NO;
          END;                              /* END UPDATE FAILED      */
        ELSE                                /* SUCCESSFUL UPDATE      */
          DO;
            CALL DSN8MPG(MODULE, '004I', OUTMSG);
            MSGS = OUTMSG;
            CALL ISPLINK(I_TBPUT, TABLE_NAME);
            IF ROWS_CHANGED > 1 THEN        /* MORE CHANGES TO DO     */
              DO;                           /* DISPLAY CHANGES        */
                CALL ISPLINK(I_TBDISPL, TABLE_NAME);
                CALL ISPLINK(I_VGET, DIS_VARS);
              END;
            ELSE                            /* NO MORE CHANGES        */
                MORE_CHANGES = NO;
          END;                              /* END SUCCESSFUL UPDATE  */
      END;                                  /* DO WHILE MORE CHANGES  */
      CALL ISPLINK(I_TBCLOSE, TABLE_NAME);     /* CLOSE ISPF TABLE    */
    END;                                       /* END IF ^DIS_EXIT    */

  END DISPLAY_LIST;

1/*********************************************************************/
 /* ERROR HANDLING                                                    */
 /*********************************************************************/

 P3_DBERROR:

  CALL DSN8MPG(MODULE, '060E', OUTMSG);   /* GET FULL MSG TEXT      */
  SQL_PIC = SQLCODE;
  MSGS = SUBSTR(OUTMSG,1,46) || SQL_PIC;  /* APPEND SQL CODE        */
  CALL ISPLINK(I_VPUT, MSGS_VAR);         /* PUT MSG OUT            */
  RETURN;                                 /* EXIT PROGRAM           */

 END DSN8SP3;
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EP1

PASS DB2 COMMANDS TO BE EXECUTED BY THE STORED PROCEDURE PROGRAM DSN8EP2.

```
 DSN8EP1: PROCEDURE(PARMS) OPTIONS(MAIN);
 /********************************************************************
  *   MODULE NAME = DSN8EP1 (SAMPLE PROGRAM)                          *
  *                                                                   *
  *   DESCRIPTIVE NAME = STORED PROCEDURE REQUESTER PROGRAM           *
  *                                                                   *
  *    LICENSED MATERIALS - PROPERTY OF IBM                           *
  *    5675-DB2                                                       *
  *    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.       *
  *                                                                   *
  *    STATUS = VERSION 7                                             *
  *                                                                   *
  *   FUNCTION =                                                      *
  *                                                                   *
  *      PASS DB2 COMMANDS TO BE EXECUTED BY THE STORED               *
  *      PROCEDURE PROGRAM DSN8EP2.  GET INPUT FROM 'SYSIN'.          *
  *      PASS THE COMMAND AND RECEIVE THE COMMAND RESULTS             *
  *      VIA THE PARAMETERS CONTAINED IN THE EXEC SQL CALL            *
  *      STATEMENT.  WRITE THE RESULTS TO 'SYSPRINT'.                 *
  *                                                                   *
  *      DEPENDENCIES = NONE                                          *
  *                                                                   *
  *      RESTRICTIONS =                                               *
  *                                                                   *
  *         1. BEGIN DB2 COMMANDS WITH A HYPHEN AND END THEM          *
  *            WITH A SEMICOLON.  A '*' IN COLUMN ONE OR '--'         *
  *            ANYWHERE ON A LINE (EXCEPT WITHIN A COMMAND) CAN       *
  *            BE USED TO DENOTE COMMENTS.                            *
  *                                                                   *
  *         2. THIS PROGRAM ACCEPTS COMMANDS OF AT MOST 4096 BYTES.   *
  *                                                                   *
  *      PROGRAM SIZES =                                              *
  *                                                                   *
  *          THE FOLLOWING VARIABLES CAN BE CHANGED TO FIT THE        *
  *          SPECIFIC ENVIRONMENT OF THE USER.                        *
```

```
*                                                                   *
*          VARIABLE   VALUE   MEANING                               *
*          NAME                                                     *
*          --------   -----   --------------------------            *
*                                                                   *
*          PAGEWIDTH   133    MAXIMUM WIDTH OF A PAGE IN            *
*                             CHARACTERS (INCLUDING THE CONTROL     *
*                             CHARACTER IN COLUMN ONE)              *
*                                                                   *
*          MAXPAGWD    125    PRINT LINE WIDTH CONTROLLER =         *
*                             MAXIMUM WIDTH - 1 (FOR CONTROL        *
*                             CHARACTER) - 6 (LENGTH OF THE         *
*                             COLUMN DISPLAY) - 1 ( A '-'           *
*                             BETWEEN THE COLUMN NUMBER DISPLAY     *
*                             THE SQL OUTPUT DISPLAY).              *
*                                                                   *
*          MAXPAGLN     60    MAXIMUM NUMBER OF LINES ON  THE       *
*                             PRINT OUTPUT PAGES 2 THRN N.  PAGE    *
*                             1  WILL HAVE MAXPAGLN + 1 LINES.      *
*                                                                   *
*          INPUTL       72    LENGTH OF THE INPUT RECORD            *
*                                                                   *
*     INPUT =                                                       *
*                                                                   *
*           1.  INPUT STATEMENTS WILL BE TRANSFERRED                *
*               TO THE STATEMENT BUFFER WITH ONE BLANK BETWEEN      *
*               WORDS.                                              *
*                                                                   *
*           2.  BLANKS IN DELIMITED STRINGS WILL BE                 *
*               TRANSFERRED INTO THE STATEMENT BUFFER               *
*               EXACTLY AS THEY APPEAR IN THE INPUT                 *
*               STATEMENT.                                          *
*                                                                   *
*           3.  AN INPUT LINE CONSISTS OF CHARACTERS FROM           *
*               COLUMNS 1-INPUTL.  IF AN INPUT STATEMENT SPANS      *
*               OVER MULITPLE LINES, THE LINES ARE CONCATENATED     *
*               AND BLANKS ARE REMOVED SUCH THAT ONLY ONE           *
*               BLANK OCCURS BETWEEN WORDS.                         *
*                                                                   *
*   MODULE TYPE  = PROCEDURE                                        *
*     PROCESSOR =                                                   *
*        ADMF PRECOMPILER                                           *
*        PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE)                   *
*     MODULE SIZE = 2K                                              *
*     ATTRIBUTES = RE-ENTERABLE                                     *
*                                                                   *
*   ENTRY POINT = DSN8EP1                                           *
*     PURPOSE  = SEE FUNCTION                                       *
*     LINKAGE  = STANDARD MVS PROGRAM INVOCATION, ONE PARAMETER.    *
*     INPUT    = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:     *
*        SYMBOLIC LABEL/NAME = SYSIN                                *
*        DESCRIPTION = DDNAME OF SEQUENTIAL DATA SET CONTAINING     *
*                      DB2 COMMANDS TO BE EXECUTED.                 *
*     OUTPUT   = PARAMETERS EXPLICITLY RETURNED:                    *
*        SYMBOLIC LABEL/NAME = SYSPRINT                             *
*        DESCRIPTION = DDNAME OF SEQUENTIAL OUTPUT DATA SET TO      *
*                      CONTAIN RESULTS OF THE COMMANDS EXECUTED.    *
*                                                                   *
*   EXIT NORMAL =                                                   *
*                                                                   *
*     NO ERRORS WERE FOUND IN THE SOURCE AND NO                     *
*     ERRORS OCCURRED DURING PROCESSING.                            *
*                                                                   *
*                                                                   *
*   NORMAL MESSAGES =                                               *
*                                                                   *
*     1. THE FOLLOWING MESSAGE WILL BE GENERATED FOR ALL INPUT      *
*        STATEMENTS:                                                *
*                                                                   *
*          ***INPUT STATEMENT:   DB2 COMMAND INPUT STATEMENT        *
*                                                                   *
*                                                                   *
*   EXIT-ERROR =                                                    *
*                                                                   *
*     ERRORS WERE FOUND IN THE SOURCE, OR OCCURRED DURING           *
*     PROCESSING.                                                   *
*                                                                   *
*     RETURN CODE =  4 - WARNING-LEVEL ERRORS DETECTED.             *
*        SQLWARNING OR IFI WARNING FOUND DURING EXECUTION.          *
*        REASON CODE = 0 OR IFI REASON CODE                         *
*                                                                   *
*     RETURN CODE =  8 - ERRORS DETECTED.                           *
```

```
*         SQLERROR OR IFI ERROR FOUND DURING EXECUTION.           *
*         REASON CODE = 0 OR IFI REASON CODE                      *
*                                                                 *
*      RETURN CODE = 12 - SEVERE ERRORS DETECTED.                 *
*       ONE OF THE FOLLOWING ERRORS OCCURRED:                     *
*         UNABLE TO OPEN FILES.                                   *
*         INTERNAL ERROR, ERROR MESSAGE ROUTINE RETURN CODE.      *
*         STATEMENT IS TOO LONG.                                  *
*         SQL OR IFI BUFFER OVERFLOW.                             *
*         REASON CODE = 0 OR IFI REASON CODE                      *
*                                                                 *
*      ABEND CODES = NONE                                         *
*                                                                 *
*   ERROR MESSAGES =                                              *
*                                                                 *
*      1. THE FOLLOWING MESSAGE WILL BE GENERATED WHEN A DB2      *
*         COMMAND DOES NOT BEGIN WITH A HYPHEN "-".               *
*                                                                 *
*            *** SYNTAX FOR DB2 COMMAND IS NOT VALID.             *
*            A VALID COMMAND MUST BEGIN WITH A HYPHEN "-".        *
*                                                                 *
*      2. THE FOLLOWING MESSAGE WILL BE GENERATED WHEN AN INPUT   *
*         STATEMENT IS GREATER THAN STMTMAX SIZE:                 *
*                                                                 *
*            **ERROR:  DB2 COMMAND GREATER THAN NNN CHARACTERS.   *
*                  STMT:                                          *
*                        DB2 COMMAND.                             *
*                                                                 *
*            NNN IS MAXIMUM COMMAND SIZE                          *
*            DB2 COMMAND IS THE CURRENT DB2 COMMAND BEING         *
*                PROCESSED.                                       *
*                                                                 *
*   EXTERNAL REFERENCES =                                         *
*      ROUTINES/SERVICES = NONE                                   *
*         DSNTIAR        - SQL COMMUNICATION AREA FORMATTING      *
*      DATA-AREAS        = NONE                                   *
*      CONTROL-BLOCKS =                                           *
*         SQLCA               - SQL COMMUNICATION AREA            *
*                                                                 *
*   PSEUDOCODE      =                                             *
*                                                                 *
*   DSN8EP1: PROCEDURE.                                           *
*   DECLARATIONS.                                                 *
*   INITIALIZE VARIABLES.                                         *
*   CALL READRTN TO READ IN A DB2 COMMAND STATEMENT.              *
*   DO UNTIL END-OF-FILE.                                         *
*      CALL READRTN TO READ A NEW DB2 COMMAND STATEMENT.          *
*   END.                                                          *
*                                                                 *
*   HEX2CHAR: PROCEDURE.                                          *
*   CONVERT THE RETURN CODE AND REASON CODE THAT ARE RETURNED     *
*     FROM THE IFI CALL FROM BINARY TO HEXADECIMAL.               *
*   END HEX2CHAR.                                                 *
*                                                                 *
*   PRINTCA: PROCEDURE.                                           *
*   CALL DSNTIAR TO FORMAT ANY MESSAGES.                          *
*   IF A RETURN CODE WAS PASSED FROM DSNTIAR, INDICATE IT.        *
*   PRINT THE DATA FORMATTED FORMATTED BY DSNTIAR.                *
*   SET THE RETURN CODE TO 8.                                     *
*   END PRINTCA.                                                  *
*                                                                 *
*   READRTN: PROCEDURE.                                           *
*   SET  ENDSTR = "NO".                                           *
*   SET  REREAD = "NO".                                           *
*   DO WHILE (ENDSTR = NO).                                       *
*     FILL THE STATEMENT BUFFER FROM THE CURRENT INPUT LINE.      *
*       AVOID INITIAL BLANKS.                                     *
*     TERMINATE A STATEMENT WHEN A SEMICOLON IS FOUND.            *
*     VERIFY THAT COMMAND IS VALID.                               *
*     DO SQL TO CALL DSN8EP2.                                     *
*     PROCESS THE COMMAND RESULTS.                                *
*       SET REREAD FLAG.                                          *
*     RETURN TO CALLER.                                           *
*     END COMMAND.                                                *
*   END READRTN.                                                  *
*                                                                 *
*   RESULTS: PROCEDURE.                                           *
*   PROCESS THE RETURN CODE, REASON CODE, THE NUMBER OF           *
*     BYTES IN THE RETURN BUFFER, AND THE RETURN BUFFER           *
*     THAT ARE RETURNED FROM THE IFI CALL.                        *
*   END RESULTS.                                                  *
*                                                                 *
```

```
 *  CHANGE ACTIVITY =                                                    *
 *   6/29/95  UPDATED THE REMOTE LOCATION NAME VARIABLES (DB2LOC @03*
 *            & PARMS) TO ACCEPT A SIXTEEN CHARACTER NAME        @03*
 *                                      (PN69303)  @03 KFF0296*
 *   7/05/95  CHANGED THE OUTPUT STRING LENGTH FROM VARYING      @35*
 *            TO FIXED 80 BYTE STRINGS        (PN72035)  @35 KFF0347*
 *   8/28/95  ADDED ROLLBACK WORK STATEMENT TO ENSURE THAT DB2   @42*
 *            WORK IS ROLLED BACK IN ERROR SITUATIONS           @42*
 *                                      (PN74842)  @42 KFF0580*
 *  04/17/00  INITIALIZE STORAGE TO PREVENT RETURN CODE=04,         *
 *            REASON CODE=00E60804 FROM IFI                 PQ36800*
 *  05/22/03  FIX CODE HOLE CLOSED BY VA AND ENTERPRISE PL/I PQ44916*
 *******************************************************************/
%PAGE;
/*******************************************************************/
/* VARIABLE DECLARATIONS                                           */
/*******************************************************************/

/*******************************************************************/
/* DECLARE IFI-RELATED VARIABLES                                   */
/*******************************************************************/
DCL
    IFCA_RET_CODE      CHAR(8)  INIT(' '), /* RETURN CODE IN HEX    */
    IFCA_RES_CODE      CHAR(8)  INIT(' '), /* REASON CODE IN HEX    */
    INPUTCMD      VAR CHAR(4096) INIT(' '),/* DB2 COMMAND           */
    IFCA_RET_HEX  FIXED BIN(31) INIT(0),   /* RETURN CODE PARAMETER */
    IFCA_RES_HEX  FIXED BIN(31) INIT(0),   /* REASON CODE PARAMETER */
    BUFF_OVERFLOW FIXED BIN(31) INIT(0),   /* BUFFER OVERFLOW IND@35*/
    REMBYTES      FIXED BIN(15) INIT(0),   /* BYTES REMAINING    @35*/
    RETURN_BUFF   VAR CHAR(8320) INIT(' '),/* COMMAND RESULT     @35*/
    RETURN_IND    FIXED BIN(15) INIT(0);   /* INDICATOR VARIABLE @35*/
                                           /* FOR RETURN_BUFFER     */


/*******************************************************************/
/* CHARACTER CONSTANTS                                             */
/*******************************************************************/

DCL
    ASTERISK    CHAR(1) INIT('*') STATIC, /* COMMENT INDICATOR     */
    BLANK       CHAR(1) INIT(' ') STATIC, /* INITIALIZATION BLANKS */
    HYPHEN      CHAR(1) INIT('-') STATIC,    /* HYPHEN             */
    NULLCHAR    CHAR(1) VAR INIT('') STATIC, /* NULL CHARACTER     */
    QUOTE       CHAR(1) INIT('''') STATIC,   /* QUOTATION MARK     */
    DQUOTE      CHAR(1) INIT('"') STATIC, /* DOUBLE QUOTATION MARK */
    SEMICOLON   CHAR(1) INIT(';') STATIC; /* SQL STMT TERMINATOR   */

/*******************************************************************/
/* PROGRAM INPUT/OUTPUT CONSTANTS                                  */
/*******************************************************************/

DCL
    INPUTL      FIXED BIN(15) INIT(72) STATIC,  /* SYSIN LRECL     */
    MAXPAGWD    FIXED BIN(31) INIT(125) STATIC, /* OUTPUT WIDTH    */
    MAXPAGLN    FIXED BIN(15) INIT(60) STATIC,  /* # LINES / PAGE  */
    OUTLEN      FIXED BIN(15) INIT(80) STATIC,  /* LENGTH OF AN @35*/
                                                /* OUTPUT LINE     */
    PAGEWIDTH   FIXED BIN(31) INIT(133) STATIC; /* SYSOUT LRECL    */
                                                /* AREA LENGTH     */

/*******************************************************************/
/* ERROR CODE CONSTANTS                                            */
/*******************************************************************/

DCL
    RETWRN      FIXED BIN(15) INIT(4) STATIC,   /* WARN RET COD @35*/
    RETERR      FIXED BIN(15) INIT(8) STATIC,   /* ERROR RET CODE  */
    SEVERE      FIXED BIN(15) INIT(12) STATIC;  /* SEVERE ERROR    */
                                                /* RETURN CODE     */

/*******************************************************************/
/* NUMBER CONSTANTS                                                */
/*******************************************************************/

DCL
    ZERO      FIXED BIN(15) INIT(0)  STATIC,
    ONE       FIXED BIN(15) INIT(1)  STATIC,
    TWO       FIXED BIN(15) INIT(2)  STATIC,
    FOUR      FIXED BIN(15) INIT(4)  STATIC,
    FIVE      FIXED BIN(15) INIT(5)  STATIC,
    EIGHT     FIXED BIN(15) INIT(8)  STATIC,
    TEN       FIXED BIN(15) INIT(10) STATIC;
```

```

/*****************************************************************/
/* FLAG CONSTANTS                                                */
/*****************************************************************/

DCL
    YES         BIT(1) INIT('1'B) STATIC,       /* BIT FLAG ON     */
    NO          BIT(1) INIT('0'B) STATIC;       /* BIT FLAG OFF    */

/*****************************************************************/
/* INPUT / OUTPUT BUFFER VARIABLES DECLARATION                   */
/*****************************************************************/

DCL
    COMMENT     BIT(1)         INIT('0'B), /* COMMENT ENCOUNTERED?   */
    CURPTR      FIXED BIN(15) INIT(0),     /* CURR LOCN IN OUTPUT @35*/
    DB2LOC2     VAR CHAR(16)  INIT(' '),   /* REMOTE DB2 LOC NAME @03*/
    ENDSTR      BIT(1)         INIT('0'B), /* END OF STATEMENT FLAG  */
    EODIN       BIT(1)         INIT('0'B), /* END OF INPUT DATA FLAG */
    ERR         FIXED BIN(15) INIT(0),     /* THE CURRENT RETURN CODE*/
    EXIT        BIT(1)         INIT('0'B), /* PROGRAM EXIT INDICATOR */
    I           FIXED BIN(15) INIT(0),     /* LOOP COUNTER VARIABLE  */
    INCOL       FIXED BIN(15) INIT(0),     /* CURRENT INPUT COLUMN   */
    INPUT(INPUTL)     CHAR(1),             /* CURRENT INPUT DATA     */
    J           FIXED BIN(15) INIT(0),     /* LOOP COUNTER VARIABLE  */
    K           FIXED BIN(15) INIT(0),     /* LOOP COUNTER VARIABLE  */
    KK          FIXED BIN(15) INIT(0),     /* LOOP COUNTER VARIABLE  */
    OSTMTLN     FIXED BIN(15) INIT(0),     /* # OF OUTPUT LINES NEED-*/
                                           /* ED FOR INPUT STATEMENT */
    PAGEBUF     VAR CHAR(15)  INIT(' '),   /* OUTPUT PAGE INFORMATION*/
    PARMS       VAR CHAR(16),              /* PROGRAM INPUT PARM  @03*/
    PRTBUF      VAR CHAR(80)  INIT(' '),   /* PRINT BUFFER        @35*/
    WRNING      BIT(1)         INIT('0'B), /* PRINT SQLCA ON WARNING */
    RETCODE     FIXED BIN(31) INIT(0);     /* RETURN CODE FOR DSN8EP1*/

/*****************************************************************/
/* BUILT IN FUNCTIONS DECLARATIONS                               */
/*****************************************************************/

DCL
    ADDR         BUILTIN,        /* FUNCTION TO RETURN THE ADDRESS */
    CHAR         BUILTIN,        /* RETURNS CHAR REPRESENTATION    */
    LENGTH       BUILTIN,        /* RETURNS LENGTH OF A STRING     */
    MIN          BUILTIN,        /* FUNCTION TO RETURN MINIMUM     */
    NULL         BUILTIN,        /* NULL VALUE                     */
    SUBSTR       BUILTIN,        /* FUNCTION TO RETURN SUBSTRING   */
    PLIRETC      BUILTIN,        /* FUNCTION TO SET RETURN CODE    */
    PLIRETV      BUILTIN,        /* PL/I RETURN CODE VALUE         */
    UNSPEC       BUILTIN;        /* IGNORES VARIABLE TYPING        */

/*****************************************************************/
/* DECLARE BUFFER AREAS FOR THE SQLCA AND THE SQLDA              */
/*****************************************************************/

EXEC SQL INCLUDE SQLCA;          /* DEFINE THE SQLCA               */

/*****************************************************************/
/* MESSAGE FORMATTING ROUTINE AND VARIABLES DECLARAIONS          */
/*****************************************************************/
DCL
    DSNTIAR ENTRY EXTERNAL OPTIONS(ASM INTER RETCODE);
DCL
    MSGBLEN     FIXED BIN(15) INIT(10);   /* MAX # SQL MESSAGES    */
DCL
    01 MESSAGE,                    /* RETURNED MESSAGES AREA      */
       02 MESSAGEL FIXED BIN(15)    /* MESSAGE BUFFER LENGTH       */
                 INIT(0),
       02 MESSAGET(MSGBLEN) CHAR(MAXPAGWD)   /* SQLCA MSGS SPACE   */
                 INIT(' ');
/*****************************************************************/
/* BUFFER DECLARATION FOR THE INPUT STATEMENT                    */
/* *** NOTE *** : THE CHARACTER SIZE MUST BE EXPLICIT FOR THE    */
/*               PRECOMPILER                                     */
/*****************************************************************/

DCL
    INPLLEN     FIXED BIN(15)  INIT(100), /* LENGTH OF PRINT STMT  */
    STMTBUF     VAR CHAR(4096) INIT(' '), /* STATEMENT STRING      */
    STMTLEN     FIXED BIN(15)  INIT(0),   /* STMT STRING LENGTH    */
    STMTMAX     FIXED BIN      INIT(4096);/* STATEMENT BUFFER      */
                                          /* MAXIMUM LENGTH        */

```

```
/*********************************************************************/
/* FILE DECLARATIONS                                                 */
/*********************************************************************/

DCL
    SYSIN       FILE STREAM INPUT,  /* INPUT FILE                    */
    SYSPRINT    FILE STREAM OUTPUT  /* OUTPUT FILE                   */
                ENV(FB,RECSIZE(PAGEWIDTH),BLKSIZE(PAGEWIDTH));
 %PAGE;
/*********************************************************************/
/* MAIN PROGRAM                                                      */
/*********************************************************************/
/* GENERAL INITIALIZATION                                            */
/*********************************************************************/

RETCODE = ZERO;                  /* INITIALIZE THE RETURN CODE     */
WRNING    = NO;                  /* INITIALIZE PRINTING SQLCA ON   */
                                 /* WARNING FLAG                   */
MESSAGEL = MSGBLEN * MAXPAGWD;   /* SET MESSAGE BUFFER LENGTH      */
DB2LOC2 = PARMS;                 /* INPUT PARAMETER IS THE REMOTE  */
                                 /* DB2 LOCATION NAME              */

/*********************************************************************/
/* INPUT PROCESSING INITIALIZATION                                   */
/*********************************************************************/

EXIT = NO;                       /* DON'T EXIT-CONTINUE PROCESSING */
EODIN = NO;                      /* NOT AT THE END OF INPUT DATA   */
INPUT = NULLCHAR;                /* NULL THE INPUT DATA ARRAY      */
INCOL = INPUTL+ONE;              /* SET COLUMN TO 73 TO INDICATE A */
                                 /* NEW LINE IS TO BE READ IN      */
                                 /* READRTN                        */

%PAGE;
/*********************************************************************/
/* READ THE FIRST COMMAND STATEMENT TO BE PROCESSED                  */
/*********************************************************************/

CALL READRTN;
/*********************************************************************/
/* MAIN LOOP.  CONTINUE PROCESSING DB2 COMMANDS  UNTIL THE END OF    */
/* DATA IS REACHED OR A SEVERE ERROR HAS BEEN ENCOUNTERED            */
/*********************************************************************/

PRC:
DO WHILE (EXIT = NO & RETCODE < SEVERE);
  ERR = ZERO;                   /* CLEAR THE CURRENT RETURN CODE   */
                                /* INCLUDE OUTPUT HEADINGS         */
  CALL READRTN;                  /* READ NEXT STATEMENT            */
END;                             /* END PRC                        */
GOTO STOPRUN;                    /* EXIT                           */

%PAGE;

HEX2CHAR:
/**************************************************/
/* PROCEDURE TO PRINT THE IFI RETURN CODE IN HEX  */
/**************************************************/
  PROCEDURE(INPUT) RETURNS(CHAR(8));    /* RESULTS RETURNED IN     */
                                        /*  CHARACTER FORMAT       */
    DECLARE INPUT BIT(31),              /* RETURN CODE IN BINARY   */
      I1 BIT(4) DEF INPUT,
      I2 BIT(4) DEF INPUT POSITION(4),
      I3 BIT(4) DEF INPUT POSITION(8),
      I4 BIT(4) DEF INPUT POSITION(12),
      I5 BIT(4) DEF INPUT POSITION(16),
      I6 BIT(4) DEF INPUT POSITION(20),
      I7 BIT(4) DEF INPUT POSITION(24),
      I8 BIT(4) DEF INPUT POSITION(28),
      HEXES CHAR(16) INIT('0123456789ABCDEF'),
      OUTPUT CHAR(8),
      OUTPUT1(8) CHAR(1) DEFINED(OUTPUT);
    OUTPUT1(1)=SUBSTR(HEXES,I1+1,1); /*1ST BYTE OF RET CODE IN HEX */
    OUTPUT1(2)=SUBSTR(HEXES,I2+1,1); /*2ND BYTE OF RET CODE IN HEX */
    OUTPUT1(3)=SUBSTR(HEXES,I3+1,1); /*3RD BYTE OF RET CODE IN HEX */
    OUTPUT1(4)=SUBSTR(HEXES,I4+1,1); /*4TH BYTE OF RET CODE IN HEX */
    OUTPUT1(5)=SUBSTR(HEXES,I5+1,1); /*5TH BYTE OF RET CODE IN HEX */
    OUTPUT1(6)=SUBSTR(HEXES,I6+1,1); /*6TH BYTE OF RET CODE IN HEX */
    OUTPUT1(7)=SUBSTR(HEXES,I7+1,1); /*7TH BYTE OF RET CODE IN HEX */
    OUTPUT1(8)=SUBSTR(HEXES,I8+1,1); /*8TH BYTE OF RET CODE IN HEX */
    RETURN (OUTPUT);                    /* RETURN THE OUTPUT RESULT*/
END HEX2CHAR;
```

```
%PAGE;
/*******************************************************************/
/* PROCEDURE TO PRINT THE SQLCA ERROR INDICATION AND CLEAR OUT THE */
/* SQLCA.  OUTPUT MOST OF THE DATA ON AN EXCEPTION BASIS           */
/*******************************************************************/

PRINTCA: PROCEDURE;

/*******************************************************************/
/* PROCESS SQL OUTPUT MESSAGE                                      */
/*******************************************************************/

CALL DSNTIAR ( SQLCA, MESSAGE, MAXPAGWD); /* FORMAT ANY MESSAGES   */
IF PLIRETV ^= ZERO THEN          /* IF THE RETURN CODE ISN'T ZERO  */
  DO;                            /* ISSUE AN ERROR MESSAGE         */
    PUT EDIT (' *** RETURN CODE ', PLIRETV,                 /*@35*/
             ' FROM MESSAGE ROUTINE DSNTIAR.')
             (COL(1), A(17), F(8), A(30));   /* ISSUE THE MESSAGE  */
    RETCODE = SEVERE;            /* SET THE RETURN CODE            */
  END;                           /* END ISSUE AN ERROR MESSAGE     */

DO I = ONE TO MSGBLEN            /* PRINT OUT THE DSNTIAR BUFFER   */
WHILE (MESSAGET(I) ^= BLANK);    /* PRINT NON BLANK LINES          */
  PUT EDIT ( MESSAGET(I) ) (COL(2), A(MAXPAGWD));
END;

RETCODE = SEVERE;                /* SET THE RETURN CODE            */

END PRINTCA;


%PAGE;

/*******************************************************************/
/* THIS PROCEDURE READS THE DATA FROM THE USER AND OBTAINS A DB2   */
/* COMMAND TO PASS TO DSN8EP2 FOR EXECUTION VIA THE IFI CALL       */
/*******************************************************************/

READRTN: PROCEDURE;

DCL
    CONTLINE    FIXED BIN(15)    /* CONTINUATION LINE - INPUT STMT */
                  INIT(0),       /* IS MORE THAN 72 CHARACTERS     */
    DQUOTFLAG   BIT(1)           /* DOUBLE QUOTE (") ENCOUNTERED?  */
                  INIT('0'B),
    FIRSTCHAR   BIT(1)           /* FIRST NON BLANK CHAR?          */
                  INIT('0'B),
    LASTCHAR    CHAR(1)          /* LAST CHARACTER IN THE BUFFER   */
                  INIT(' '),
    MOVECHAR    BIT(1)           /* MOVE CHAR INTO STMT BUFFER?    */
                  INIT('0'B),
    NBLK        FIXED BIN(15)    /* NUMBER OF BLANKS FOUND         */
                  INIT( 0 ),
    NEWOFSET    FIXED BIN(15)    /* FIRST POSITION OF THE COMMAND  */
                  INIT( 0 ),     /* IN THE STATEMENT BUFFER        */
    NEWSTMT     BIT(1)           /* NEW STMT TO BE PROCESSED?      */
                  INIT('0'B),
    QUOTEFLAG   BIT(1)           /* QUOTE (') ENCOUNTERED?         */
                  INIT('0'B);

/*******************************************************************/
/* ENDFILE CONDITIONS                                              */
/*******************************************************************/

ON ENDFILE(SYSIN)                /* PROCESS EOF ON INPUT FILE      */
  BEGIN;                         /* END OF FILE                    */
    IF LENGTH(STMTBUF) = 0 THEN
      DO;                        /* LENGTH(STMTBUF) = 0            */
        EXIT = YES;              /* NO STMT TO PROCESS,            */
        GOTO ENDRD;              /*  SO END THE PROGRAM            */
      END;                       /* END LENGTH(STMTBUF) = 0        */
    ELSE                         /* PROCESS THE CURRENT STATEMENT  */
      DO;                        /* LENGTH(STMTBUF) ^= 0           */
        EODIN = YES;             /* SIGNAL END_OF_DATA             */
        ENDSTR = YES;            /* SIGNAL END_OF_STRING           */
        GOTO CHKCOMM;            /* PROCESS CURRENT COMMAND        */
      END;                       /* END LENGTH(STMTBUF) ^= 0       */
  END;                           /* END END OF FILE                */

/*******************************************************************/
/* BEGIN READRTN PROCESSING                                        */
```

```
/*******************************************************************/

NEWSTMT= YES;                    /* NEW STMT IS BEING PROCESSED   */

%PAGE;

/*******************************************************************/
/* READ IN THE INPUT STATEMENT                                     */
/*******************************************************************/

RD:

DO WHILE (NEWSTMT = YES);

  /*****************************************************************/
  /* NO MORE INPUT DATA (EOF) SO RETURN TO CALLER                  */
  /*****************************************************************/

  IF EODIN = YES THEN
    DO;                          /* END OF DATA                    */
      EXIT = YES;                /* EXIT PROGRAM                   */
      LEAVE RD;                  /* LEAVE THE LOOP                 */
    END;                         /* END END OF DATA                */

  /*****************************************************************/
  /* PROCESS THE STATEMENT                                         */
  /*****************************************************************/

  ELSE                           /* MORE INPUT TO PROCESS          */
    DO;
      NEWSTMT = NO;              /* TURN NEW STATEMENT FLAG OFF    */
      CONTLINE = ZERO;           /* CLEAR MULTILINE STMT COUNTER   */
      ENDSTR = NO;               /* NOT AT THE END OF THE STRING   */
      QUOTEFLAG = NO;            /* INITIALIZE QUOTE FLAG          */
      DQUOTFLAG = NO;            /* INITIALIZE DOUBLE QUOTE FLAG   */
      STMTLEN = ZERO;            /* INITIALIZE THE STMT LENGTH     */
      STMTBUF = NULLCHAR;        /* INIT STMT BUFFER TO NULLS      */
      LASTCHAR = NULLCHAR;       /* INIT. LAST CHARACTER TO NULL   */
      COMMENT = NO;              /* INITIALIZE THE COMMENT FLAG    */
      FIRSTCHAR = NO;            /* INIT. FIRST CHAR TO NO         */
      NBLK = ZERO;               /* INIT. BLANK COUNT TO 0         */

      /*************************************************************/
      /* READ AND PROCESS A NEW STATEMENT                          */
      /*************************************************************/

      DO WHILE (ENDSTR = NO);    /* PUT INPUT STMT IN STMT BUFFER  */

        /***********************************************************/
        /* IF THE COLUMN BEING PROCESSED IS GREATER THAN THE       */
        /* LENGTH OF THE INPUT LINE THEN READ THE NEXT LINE        */
        /***********************************************************/

        IF INCOL > INPUTL THEN
          DO;                                     /* GET SYSIN DATA */
            GET EDIT (INPUT) (COL(1), (INPUTL) A(1));  /*           */
            INCOL = ONE;               /* POINT TO FIRST CHARACTER */
            IF FIRSTCHAR = YES THEN    /* FIRST CHAR SET?          */
              CONTLINE = CONTLINE + 1; /* INCREMENT INPUT LINE CTR */
          END;

        /***********************************************************/
        /* THE CHARACTER IN COLUMN ONE IS AN ASTERISK OR THE       */
        /* CHARACTERS IN COLUMNS 1 AND 2 ARE '--'.  CONSIDER THIS  */
        /* LINE TO BE A COMMENT.  PRINT THE LINE AND RETRIEVE THE  */
        /* NEXT INPUT LINE.                                        */
        /***********************************************************/

        IF INCOL = 1 & (INPUT(1) = ASTERISK
              | (INPUT(1) = HYPHEN & INPUT(2) = HYPHEN))
                & STMTLEN = 0 THEN
          DO;                    /* STATEMENT IS A COMMENT         */
            DO J = 1 TO INPUTL;  /* PUT ENTIRE LINE INTO STMTBUF   */
              STMTBUF = STMTBUF || INPUT(J);
            END;
            STMTLEN = LENGTH(STMTBUF);
            ENDSTR = YES;           /* INDICATE END OF A STRING    */
            NEWSTMT = YES;          /* NEW STMT SHOULD BE READ     */
            INCOL = INPUTL + ONE;   /* SET INDEX TO 73 TO FORCE    */
                                    /* THE NEXT STMT TO BE READ    */
            COMMENT= ^COMMENT;      /* SET COMMENT INDICATOR ON    */
          END;                      /* END STATEMENT IS A COMMENT  */
```

```
/***********************************************************/
/* PROCESS THE INPUT STATEMENT                             */
/***********************************************************/

ELSE
  DO;

    /*******************************************************/
    /* MOVE THE CHARACTER FROM THE INPUT DATA INTO THE     */
    /* STATEMENT BUFFER UNTIL AN END OF LINE CHARACTER     */
    /* OR SEMICOLON IS ENCOUNTERED                         */
    /*******************************************************/

    DO J = INCOL TO INPUTL WHILE (^ENDSTR);

      /*****************************************************/
      /* PREPROCESS ANY DOUBLE QUOTATION MARKS ("). IF THE */
      /* DOUBLE QUOTATION MARK IS CONTAINED BETWEEN        */
      /* QUOTATION MARKS ('), THE QUOTATION MARK IS        */
      /* CONSIDERED TO BE THE STRING DELIMITER.  THE       */
      /* DQUOTFLAG WILL NOT BE SET. IN THIS CASE THE       */
      /* DOUBLE QUOTATION MARK IS CONSIDERED TO BE PART OF */
      /* THE STRING                                        */
      /*****************************************************/

      IF INPUT(J) = DQUOTE THEN
        DO;                     /* INPUT(J)=DQUOTE         */
          IF ^QUOTEFLAG THEN    /* NOT DELIMITED BY QUOTES */
                                          /* THEN DOUBLE   */
                                          /* QUOTES ARE    */
                DQUOTFLAG = ^DQUOTFLAG;   /* THE DELIMITER */
        END;                    /* END INPUT(J) = DQUOTE   */

      /*****************************************************/
      /* PREPROCESS ANY QUOTATION MARKS (').  IF THE       */
      /* QUOTATION MARK IS CONTAINED BETWEEN DOUBLE        */
      /* QUOTATION MARKS ("), THE DOUBLE QUOTATION MARK IS */
      /* CONSIDERED TO BE THE STRING DELIMITER.  THE       */
      /* QUOTEFLAG WILL NOT BE SET. IN THIS CASE THE       */
      /* QUOTATION MARK IS CONSIDERED TO BE PART OF THE    */
      /* STRING.                                           */
      /*****************************************************/

      IF INPUT(J) = QUOTE THEN
        DO;                       /* INPUT(J) = QUOTE      */
          IF ^DQUOTFLAG THEN      /* NOT DELIMITED BY      */
                                  /* DOUBLE QUOTES THEN    */
                                  /* SINGLE QUOTES ARE THE */
              QUOTEFLAG = ^QUOTEFLAG;   /* DELIMITER       */
        END;                      /* END INPUT(J) = QUOTE  */

      /*****************************************************/
      /* PROCESS A HYPHEN IF FOUND.  THE HYPHEN IS         */
      /* CONSIDERED PART OF A STRING IF A DELIMITER FLAG   */
      /* IS SET.  IF THE FOLLOWING CHARACTER IS A HYPHEN,  */
      /* MOVE THE REMAINING CHARACTERS TO THE STATEMENT    */
      /* BUFFER.                                           */
      /*****************************************************/

      IF (INPUT(J) = HYPHEN) & /*INPUT CHAR IS '-'         */
       (J < INPUTL)          & /* STILL MORE &             */
       ^QUOTEFLAG            & /* NOT CURRENTLY IN         */
       ^DQUOTFLAG THEN         /* DELIMITED STRING THEN    */
        DO;                    /* LOOK FOR '--'            */
          IF INPUT(J+1) = HYPHEN THEN  /* FOUND '--'       */
            DO;                /* DO NOT MOVE CHARACTERS   */
            MOVECHAR = NO;     /* INTO THE STATEMENT BUFFER*/
            END;
          IF (INPUT(J+1) = HYPHEN) &
           (MOVECHAR = NO) THEN   /* COMMENT FOUND         */
            DO;                   /* STATEMENT IS A COMMENT*/
              DO J = 1 TO INPUTL;
              STMTBUF = STMTBUF || INPUT(J);
              END;       /* PUT ENTIRE LINE INTO STMTBUF   */
              STMTLEN = LENGTH(STMTBUF);
              ENDSTR = YES;    /* INDICATE END OF A STRING */
              NEWSTMT = YES;   /* NEW STMT SHOULD BE READ  */
              INCOL = INPUTL + ONE;   /* SET INDEX TO 73   */
                            /* TO FORCE THE NEXT STATEMENT */
                            /* TO BE READ                  */
              COMMENT= ^COMMENT;      /* SET THE COMMENT   */
```

```
                                /* INDICATOR ON      */
      END;           /* END STATEMENT IS A COMMENT  */
  END;               /* END LOOK FOR '--'           */
/*****************************************************/
/* PROCESS THE END-OF-STRING IF A SEMICOLON IS       */
/* FOUND.  THE SEMICOLON CANNOT BE CONTAINED WITHIN  */
/* A DELIMITED STRING.  THE ACCEPTABLE DELIMITERS    */
/* ARE QUOTE OR DOUBLE QUOTE MARKS.                  */
/*****************************************************/

IF (INPUT(J) = SEMICOLON) & ^DQUOTFLAG &
 ^QUOTEFLAG THEN         /* SEMICOLON & NOT          */
   ENDSTR = ^ENDSTR;     /* DELIMITED THEN SET END   */
                         /* OF STRING                */
/*****************************************************/
/* NOT THE END OF THE STRING, PROCESS THE STATEMENT  */
/*****************************************************/

ELSE
  DO;

  /***************************************************/
  /* MOVE ALL NON BLANK CHARACTERS INTO THE DB2      */
  /* COMMAND STATEMENT BUFFER                        */
  /***************************************************/

    IF INPUT(J)^= BLANK THEN
      DO;
        MOVECHAR = YES;
        FIRSTCHAR = YES;
        NBLK = ZERO;
      END;

  /***************************************************/
  /* A BLANK SHOULD BE MOVED IN THE FOLLOWING CASES: */
  /*                                                 */
  /*    1.  IF THE BLANK IS IN A DELIMITED STRING    */
  /*                                                 */
  /*    2.  IF AN INPUT STATEMENT SPANS MORE THAN    */
  /*        ONE LINE AND THE PREVIOUS LINE HAD A     */
  /*        CHARACTER IN COLUMN 72 AND THE CURRENT   */
  /*        LINE HAS BLANKS BEFORE THE FIRST WORD    */
  /***************************************************/

    ELSE           /* BLANK CHARACTER FOUND          */
      DO;
        IF QUOTEFLAG | DQUOTFLAG |
           (CONTLINE >= 1 & J = 1 & NBLK = 0) THEN
          DO;            /* BLANK IS DELIMITED, MOVE */
            MOVECHAR = YES;    /* IT INTO STMT BUFFER*/
            NBLK = NBLK + ONE; /* & INC BLANK COUNT  */
          END;
        ELSE                 /* BLANK NOT DELIMITED  */
          DO;
            NBLK = NBLK + ONE; /* INCREASE BLANK CTR */
            IF (NBLK = ONE) & (FIRSTCHAR = YES) THEN
              MOVECHAR = YES;
            ELSE
              DO;
              MOVECHAR = NO;
              END;
          END;        /* END BLANK NOT DELIMITED     */
      END;            /* END BLANK CHARACTER FOUND   */

    /*************************************************/
    /* IF MOVECHAR IS SET THEN MOVE THE INPUT        */
    /* CHARACTER INTO STATEMENT BUFFER AREA          */
    /*************************************************/

    IF MOVECHAR = YES THEN
      DO;

        /*********************************************/
        /* WHEN THE STATEMENT LENGTH IS TOO LONG,THE */
        /* STATEMENT CANNOT BE PROCESSED.  A RETURN  */
        /* CODE IS SET TO INDICATE NO FURTHER        */
        /* PROCESSING SHOULD BE DONE.  AN ERROR      */
        /* MESSAGE WILL BE PUT OUT.                  */
        /*********************************************/

        STMTLEN = LENGTH(STMTBUF);
        IF STMTLEN = STMTMAX THEN /* STMT TOO LONG   */
```

```
                        DO;
                          RETCODE = SEVERE;     /* SET RETURN CODE */
                          PUT EDIT(' *** ERROR:  STATEMENT GREATER ',
                                   'THAN ',STMTMAX,' CHARACTERS. ',
                                   'STMT:  ')            /*     @35*/
                                   (COL(1),A(31),A(5),F(4),A(13),
                                   A(7));                /*     @35*/
                          PUT EDIT((SUBSTR(STMTBUF,KK,
                                    MIN(100,STMTLEN-KK+1))
                                    DO KK = 1 TO STMTLEN BY 100))
                                    (COL(2),A(100));     /*     @35*/
                          LEAVE RD;
                        END;     /* END STMT TOO LONG              */
                      STMTBUF = STMTBUF || INPUT(J);
                    END;         /* MOVE CHARACTER INTO BUFFER     */
                  LASTCHAR = INPUT(J); /* SAVE THIS CHARACTER      */
                END;             /* END CHARACTER NOT A SEMICOLON  */
              END;               /* END DO J = INCOL TO INPUTL     */
          END;                   /* END PROCESS THE INPUT STMT     */
        INCOL = J;               /* UPDATE THE INPUT COLUMN        */
      END;                       /* END DO WHILE (ENDSTR = NO)     */

      /**************************************************************/
      /* CHECK WHETHER THE COMMAND ENTERED IS A COMMENT.  IF NOT,  */
      /* PRINT THE DB2 COMMAND INPUT STATEMENT.                     */
      /**************************************************************/
CHKCOMM:
      IF ^COMMENT THEN
        DO;
          STMTLEN = LENGTH(STMTBUF);
          NEWOFSET = ONE;
        END;
                /****************************************************/
                /* PRINT OUT THE DB2 COMMAND INPUT STATEMENT      */
                /****************************************************/
     PUT SKIP;
     IF ^COMMENT THEN
       DO;
         PUT SKIP;                                            /*@35*/
         PUT EDIT (' *** INPUT STATEMENT:  ') (COL(1), A);    /*@35*/
         J = STMTLEN;                                         /*@35*/
         PUT EDIT ((SUBSTR(STMTBUF,KK,MIN(INPLLEN,J-KK+1))
           DO KK = 1 TO STMTLEN BY INPLLEN))
           (A(INPLLEN),COL(1));
       END;
     ELSE                                                     /*@35*/
       DO;                                                    /*@35*/
         J = STMTLEN;                                         /*@35*/
         PUT EDIT ((SUBSTR(STMTBUF,KK,MIN(INPLLEN,J-KK+1))    /*@35*/
           DO KK = 1 TO STMTLEN BY INPLLEN))                  /*@35*/
           (COL(2),A(INPLLEN),COL(1));                        /*@35*/
       END;                                                   /*@35*/
     IF ^COMMENT THEN
         STMTBUF = SUBSTR(STMTBUF,ONE,STMTLEN);
                  /**************************************************/
                  /* UPDATE THE OUTPUT LINE COUNTER               */
                  /**************************************************/

         OSTMTLN = STMTLEN/INPLLEN; /* # LINES NEEDED FOR */
                                    /* INPUT STMT         */
         IF OSTMTLN * INPLLEN ^= STMTLEN THEN
             OSTMTLN = OSTMTLN + ONE;

         /******************************************************/
         /* CHECK THAT THE DB2 COMMAND BEGINS WITH A HYPHEN.  */
         /* IF NOT, CALL BADCMD AND ISSUE AN ISSUE AN ERROR   */
         /* MESSAGE.                                          */
         /******************************************************/

         IF ^COMMENT THEN
           DO;                          /* STATEMENT NOT A COMMENT */
/********************************************************************/
/* HANDLE BAD IFI CALL SYNTAX                                       */
/********************************************************************/
             IF SUBSTR(STMTBUF,ONE,ONE) ^= '-' THEN   /* NO HYPHEN */
               DO;
                 PUT SKIP;
                 PUT SKIP EDIT(' *** SYNTAX FOR DB2 COMMAND ',/*@35*/
                               'IS NOT VALID.')               /*@35*/
                              (COL(1),A(28),A(13));           /*@35*/
                 PUT SKIP EDIT(' *** A VALID COMMAND MUST ',  /*@35*/
                               'BEGIN WITH A HYPHEN.')
```

```
                        (COL(1),A(26),A(20));          /*@35*/
                 RETCODE = RETERR;           /* SET RET CODE TO 8 */
               END;                              /* END NO HYPHEN */
/*****************************************************************/
/* COMMAND SYNTAX IS CORRECT                                     */
/*****************************************************************/
             ELSE
               DO;                                    /* A VALID */
                 INPUTCMD = SUBSTR(STMTBUF,ONE,STMTLEN); /* COMMAND*/
                                                 /*SO MAKE CALL*/
               /**************************************************/
               /* CONNECT TO THE DB2 REMOTE LOCATION             */
               /**************************************************/
                 EXEC SQL CONNECT TO :DB2LOC2;  /* CONNECT TO      */
                                                /* REMOTE LOCATION */
                 IF SQLCODE < 0 THEN            /* SQL ERROR?   @42*/
                   DO;                          /* YES, ERROR FOUND*/
                     PUT EDIT (' *** CONNECTION TO ',DB2LOC2, /*@35*/
                               ' NOT SUCCESSFUL:')
                              (COL(1), A(19), A(16), A(16));  /*@35*/
                     CALL PRINTCA;              /* PRINT ERROR MSG */
                     GOTO STOPRUN;              /* END PROGRAM     */
                   END;                         /* END ERROR FOUND */
               /**************************************************/
               /* CALL THE STORED PROCEDURE PROGRAM DSN8EP2      */
               /**************************************************/
                 RETURN_IND = -1;                             /*@35*/
                 EXEC SQL CALL DSN8.DSN8EP2(:INPUTCMD,
                                 :IFCA_RET_HEX,
                                 :IFCA_RES_HEX,
                                 :BUFF_OVERFLOW,              /*@35*/
                                 :RETURN_BUFF:RETURN_IND);    /*@35*/
                 IF SQLCODE < 0 THEN            /* SQL ERROR?   @42*/
                   DO;                          /* YES ERROR FOUND */
                    PUT EDIT (' *** CALL TO DSN8EP2 NOT SUCCESSFUL:')
                              (COL(1),A(36));                 /*@35*/
                    IF SQLCODE = -911 | SQLCODE = -918        /*@42*/
                     | SQLCODE = -919 | SQLCODE = -965        /*@42*/
                     THEN        /* CHECK FOR SPECIFIC ERRORS   @42*/
                                 /* THAT REQUIRE A ROLL BACK    @42*/
                      DO;        /* YES, ROLL BACK REQUIRED     @42*/
                        CALL PRINTCA;      /* PRINT ERROR MSG   @42*/
                        PUT EDIT (' *** ISSUE ROLLBACK WORK ',
                                  'BECAUSE STORED PROCEDURE ',
                                  'CALL NOT SUCCESSFUL')
                                 (COL(1), A(25), A(25), A(19));
                                 /* PRINT ROLLBACK WORK MESSAGE @42*/
                        EXEC SQL ROLLBACK WORK; /* EXECUTE ROLLBACK*/
                                                /*  WORK STMT   @42*/
                      END;       /* END ROLL BACK REQUIRED      @42*/
                    CALL PRINTCA;               /* PRINT ERROR MSG */
                    GOTO STOPRUN;               /* END PROGRAM     */
                   END;                         /* END ERROR FOUND */
/*****************************************************************/
/* CALL THE RESULTS PROC TO PROCESS THE RETURN CODE, THE REASON  */
/* CODE AND THE RESULTS MESSAGE OF THE COMMAND EXECUTED BY IFI.  */
/* NEXT, INITIALIZE THE VARIABLES TO PROCESS THE NEXT DB2 COMMAND. */
/*****************************************************************/
                 CALL RESULTS;            /* PROCESS THE RESULTS   */
               END;                       /* END VALID COMMAND     */
             NEWOFSET = ZERO;             /* RESET CHARACTER PTR   */
             NEWSTMT   = YES;             /* RESET FOR NEW STMT    */
           END;                     /* END STATEMENT NOT A COMMENT */
    END;                                  /* END ELSE MORE INPUT   */
END;                                      /* END DO WHILE NEW STMT */

ENDRD:;                                   /* END RD SUB-PROC       */
END READRTN;                              /* END READRTN PROC      */

%PAGE;

/*****************************************************************/
/* PROCESS THE DB2 COMMAND RESULTS FROM THE IFCA RETURN BUFFER   */
/*****************************************************************/
RESULTS: PROCEDURE;
DCL
  M0LENGTH         CHAR(2)  INIT(' '),    /* LENGTH OF CMD RESULT  */
  M1LENGTH          BIT(16) INIT('0'B),   /* INTERNALLY STORED LNG */
  M2LENGTH    FIXED BIN(15) INIT(0),      /* LENGTH OF MESSAGE N   */
  BEGINSTR    FIXED BIN(15) INIT(1),      /* CHAR 1 POINTER */
  TOTBYTES    FIXED BIN(31) INIT(0);      /* MSG BYTE COUNT */

```

```
  IFCA_RET_CODE = HEX2CHAR(IFCA_RET_HEX);  /* RETURN CODE IN HEX   */
  IFCA_RES_CODE = HEX2CHAR(IFCA_RES_HEX);  /* REASON CODE IN HEX   */
  TOTBYTES = 0;                            /* INITIALIZE COUNTER   */
  BEGINSTR = 1;                            /* INITIALIZE POINTER   */

  IF IFCA_RET_HEX ^= 0 THEN       /* IF THE RETURN CODE ISN'T ZERO */
                                  /* ISSUE AN ERROR MESSAGE        */
    DO;
      PUT EDIT(' *** RETURN CODE=',SUBSTR(IFCA_RET_CODE,7,2), /*@35*/
      ' REASON CODE=',IFCA_RES_CODE,' FROM IFI REQUEST')
      (COL(1),A(17),A(2),A,A(8),A);                          /*@35*/
    END;                          /* END ISSUE AN ERROR MESSAGE    */

  IF LENGTH(RETURN_BUFF) ^= 0 THEN                           /*@35*/
                            /* DON'T PRINT UNLESS SOME DATA RET. */
    DO;
      PUT SKIP;                                              /*@35*/
      PUT SKIP EDIT(' *** IFI RETURN AREA:')                 /*@35*/
      (COL(1),A);                                            /*@35*/
      /***********************************************************/
      /* PROCESS THE UNFORMATTED COMMAND RESULTS FROM THE IFI CALL.*/
      /* GET THE LENGTH OF EACH RESULT LINE FROM THE FIRST TWO     */
      /* BYTES.  PUT IT IN USABLE FORM.  PRINT THE RESULTS FROM    */
      /* THE FIRST LINE.  UPDATE THE POINTER AND THE COUNTERS AND  */
      /* REPEAT UNTIL ALL BYTES FROM IFCA_BYTES_MOVED HAVE BEEN    */
      /* PROCESSED.                                                */
      /***********************************************************/
      CURPTR = 0;                 /* START OF DATA IN RET AREA@35*/
      REMBYTES = LENGTH(RETURN_BUFF); /* NUMBER OF BYTES TO PROC@35*/
      DO WHILE (REMBYTES > 0);    /* RETURN AREA PRINT LOOP   @35*/
        PRTBUF = SUBSTR(RETURN_BUFF,CURPTR,OUTLEN);          /*@35*/
        SUBSTR(PRTBUF,1,1) = BLANK;   /* BLANK FIRST COLUMN TO  @35*/
                                  /* AVOID CARRIAGE CTRL PROB  */
        PUT SKIP EDIT (PRTBUF) (COL(1),A(OUTLEN));           /*@35*/
        CURPTR = CURPTR + OUTLEN;                            /*@35*/
        REMBYTES = REMBYTES - OUTLEN;                        /*@35*/
      END;                                                   /*@35*/
    END;                              /* END IFCA_BYTES_MOVED ^= 0 */

  IF BUFF_OVERFLOW = 1 THEN           /* COULDN'T GET ALL DATA  @35*/
    DO;                                                      /*@35*/
      PUT SKIP EDIT (' *** INSUFFICIENT SPACE TO RECEIVE ',  /*@35*/
        'ALL OUTPUT FROM IFI RETURN AREA.')                  /*@35*/
        (A(35),A(32));                                       /*@35*/
      IF RETCODE < RETWRN THEN                               /*@35*/
        RETCODE = RETWRN;                                    /*@35*/
    END;                                                     /*@35*/
  IF IFCA_RET_HEX > RETCODE THEN          /* CHECK RETURN CODES    */
    RETCODE = IFCA_RET_HEX;               /* USE THE HIGHEST ONE   */

  IF IFCA_RET_HEX = SEVERE THEN           /* IF RETURN CODE = 12   */
        GOTO STOPRUN;                     /* STOP PROGRAM EXECUTION*/

END RESULTS;                              /* END RESULTS PROC      */
/*****************************************************************/
/* SET THE PL/I RETURN CODE AND TERMINATE PROCESSING             */
/*****************************************************************/

STOPRUN:
IF RETCODE >= SEVERE THEN                                    /*@35*/
  DO;                                                        /*@35*/
    PUT SKIP;                                                /*@35*/
    PUT SKIP EDIT (' *** SEVERE ERROR OCCURRED. ',           /*@35*/
      'PROGRAM IS TERMINATING.')                             /*@35*/
      (A(28),A(23));                                         /*@35*/
  END;                                                       /*@35*/
CALL PLIRETC(RETCODE);                   /* SET PLI RETURN CODE    */
END DSN8EP1;                             /* END PROGRAM            */
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8EP2

USING THE INSTRUMENTATION FACILITY INTERFACE (IFI), PROCESS A Db2 COMMAND WHICH IS PASSED FROM THE DSN8EP1 REQUESTER PROGRAM.

```
DSN8EP2:   PROCEDURE(INPUTCMD, IFCA_RET_HEX, IFCA_RES_HEX,
           BUFF_OVERFLOW, RETURN_BUFF, NULL_ARRAY)
           OPTIONS(MAIN NOEXECOPS);

/*******************************************************************
 *   MODULE NAME = DSN8EP2 (SAMPLE PROGRAM)                        *
 *                                                                 *
 *   DESCRIPTIVE NAME = STORED PROCEDURE SERVER PROGRAM            *
 *                                                                 *
 *    LICENSED MATERIALS - PROPERTY OF IBM                         *
 *    5675-DB2                                                     *
 *    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.     *
 *                                                                 *
 *    STATUS = VERSION 7                                           *
 *                                                                 *
 *   FUNCTION =                                                    *
 *      USING THE INSTRUMENTATION FACILITY INTERFACE (IFI),        *
 *      PROCESS A DB2 COMMAND WHICH IS PASSED FROM THE DSN8EP1     *
 *      REQUESTER PROGRAM.                                         *
 *                                                                 *
 *   NOTES =                                                       *
 *      DEPENDENCIES = NONE                                        *
 *                                                                 *
 *      RESTRICTIONS =                                             *
 *         1.  THE INSTRUMENTATION FACILITY COMMUNICATION AREA     *
 *             (IFCA) CONTAINS INFORMATION REGARDING THE SUCCESS   *
 *             OF THE CALL AND PROVIDES FEEDBACK.                  *
 *             THIS AREA MUST BE MAINTAINED TO INCLUDE ANY CHANGES *
 *             TO THE MAPPING MACRO DSNDIFCA.                      *
 *                                                                 *
 *   MODULE TYPE  = PROCEDURE                                      *
 *      PROCESSOR =                                                *
 *         ADMF PRECOMPILER                                        *
 *         PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE)                *
 *      MODULE SIZE = 1K                                           *
 *      ATTRIBUTES = RE-ENTERABLE                                  *
 *                                                                 *
 *   ENTRY POINT = DSN8EP2                                         *
 *      PURPOSE  = SEE FUNCTION                                    *
 *      LINKAGE  = INVOKED VIA EXEC SQL CALL.                      *
 *      INPUT    = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:  *
 *         SYMBOLIC LABEL/NAME = INPUTCMD                          *
 *         DESCRIPTION = DB2 COMMAND TO BE PROCESSED BY IFI.       *
 *                       INPUT STATEMENTS FROM THE INPUTCMD        *
 *                       PARAMETER WILL BE PASSED TO THE           *
 *                       TEXT_OR_COMMAND FIELD OF THE OUTPUT_AREA  *
 *                       IN THE DSN8EP1 PROGRAM.                   *
 *                                                                 *
 *      OUTPUT   = PARAMETERS EXPLICITLY RETURNED:                 *
 *         SYMBOLIC LABEL/NAME = IFCA_RET_HEX                      *
 *         SYMBOLIC LABEL/NAME = IFCA_RES_HEX                      *
 *         SYMBOLIC LABEL/NAME = IFCA_BYTES_MOVED                  *
 *         DESCRIPTION = COMMUNICATION AREAS BETWEEN THE           *
 *                       APPLICATION PROGRAM AND IFI               *
 *         SYMBOLIC LABEL/NAME = RETURN_BUFF                       *
 *         DESCRIPTION = DB2 COMMAND RESPONSE FROM IFI             *
 *                       THE RETURN CODE, REASON CODE, AND THE     *
 *                       BYTES MOVED FROM THE IFCA AND THE         *
 *                       RTRN_BUFF FIELD FROM THE IFI RETURN AREA  *
 *                       WILL BE PASSED VIA THE IFCA_RET_HEX,      *
 *                       IFCA_RES_HEX, IFCA_BYTES_MOVED, AND       *
 *                       RETURN_BUFF PARAMETERS.                   *
 *                                                                 *
 *   EXIT NORMAL =                                                 *
 *      NO ERRORS WERE FOUND IN THE SOURCE AND NO                  *
 *      ERRORS OCCURRED DURING PROCESSING.                         *
 *                                                                 *
 *   NORMAL MESSAGES =                                             *
 *                                                                 *
 *   EXIT-ERROR =                                                  *
 *      ERRORS WERE FOUND IN THE SOURCE, OR OCCURRED DURING        *
 *      PROCESSING.                                                *
```

```
 *                                                                    *
 *      RETURN CODE =  4 - WARNING-LEVEL ERRORS DETECTED.             *
 *         WARNING FOUND DURING EXECUTION.                            *
 *         REASON CODE = NONE OR FROM IFI                             *
 *                                                                    *
 *      RETURN CODE =  8 - ERRORS DETECTED.                           *
 *         ERROR FOUND DURING EXECUTION.                              *
 *         REASON CODE = NONE OR FROM IFI                             *
 *                                                                    *
 *      RETURN CODE = 12 - SEVERE ERRORS DETECTED.                    *
 *         UNABLE TO OPEN FILES.                                      *
 *         INTERNAL ERROR, ERROR MESSAGE ROUTINE RETURN CODE.         *
 *         STATEMENT IS TOO LONG.                                     *
 *         BUFFER OVERFLOW.                                           *
 *         REASON CODE = NONE OR FROM IFI                             *
 *                                                                    *
 *      ABEND CODES = NONE                                            *
 *                                                                    *
 *   ERROR MESSAGES =                                                 *
 *                                                                    *
 *   EXTERNAL REFERENCES =                                            *
 *      ROUTINES/SERVICES = NONE                                      *
 *      DATA-AREAS        = NONE                                      *
 *      CONTROL-BLOCKS    = NONE                                      *
 *                                                                    *
 *   PSEUDOCODE      =                                                *
 *     DSN8EP2: PROCEDURE.                                            *
 *     GET THE RETURN AREA SIZE FOR COMMAND REQUESTS.                 *
 *     ALLOCATE THE REQUESTED RETURN AREA.                            *
 *     FORMAT THE OUTPUT AREA WITH THE REQUESTED COMMAND.             *
 *     ISSUE COMMAND REQUEST.                                         *
 *     PASS RESULTS TO THE OUTPUT PARAMETERS.                         *
 *                                                                    *
 *  CHANGE ACTIVITY =                                                 *
 *   7/05/95  CHANGED THE OUTPUT STRING LENGTH FROM VARYING           *
 *            TO FIXED 80 BYTE STRINGS          PN72035  @47 KFF0347*
 *  04/17/00  INITIALIZE STORAGE TO PREVENT RETURN CODE=04,           *
 *            REASON CODE=00E60804 FROM IFI                  PQ36800*
 *  05/22/03  FIX CODE HOLE CLOSED BY VA AND ENTERPRISE PL/I PQ44916*
 *********************************************************************/
%PAGE;
/*********************************************************************/
/* VARIABLE DECLARATIONS                                             */
/*********************************************************************/

DCL COMMAND     CHAR(8) INIT(' '); /* USER SPECIFIED DB2 COMMAND    */
/*********************************************************************/
/* BUILT-IN VARIABLES                                             @47*/
/*********************************************************************/

 DCL                                                            /*@47*/
    ADDR         BUILTIN,        /* ADDRESS OF A DATA AREA       @47*/
    LENGTH       BUILTIN,        /* RETURNS LENGTH OF A STRING   @47*/
    MOD          BUILTIN,        /* RETURNS MODULO VALUE         @47*/
    STORAGE      BUILTIN,        /* FUNCTION TO GET SOME SPACE   @47*/
    SUBSTR       BUILTIN,        /* FUNCTION TO RETURN SUBSTRING @47*/
    UNSPEC       BUILTIN;        /* IGNORES VARIABLE TYPING      @47*/
/************************************************************/
/*       DECLARATION FOR INPUT AND OUTPUT PARAMETERS       */
/************************************************************/

  DCL
   FUNCTION          CHAR(8) INIT(' '),     /* FUNC PARM FOR IFI @47*/
   INPUTCMD          CHAR(4096) VARYING,    /* DB2 COMMAND       @47*/
   IFCA_RET_HEX      FIXED BIN(31),         /* IFI RETURN CODE   @47*/
   IFCA_RES_HEX      FIXED BIN(31),         /* IFI REASON CODE   @47*/
   BUFF_OVERFLOW     FIXED BIN(31),         /* RETURN BUFF BYTES @47*/
                                            /* RETURNED FROM CALL   */
   NULL_ARRAY(5)     FIXED BIN(15),         /* INDICATOR ARRAY   @47*/
   RETURN_BUFF       CHAR(8320) VARYING,    /* PASSED BUFFER     @47*/
   RETURN_LEN        FIXED BIN(15) INIT(8320) STATIC;  /* LENGTH @47*/
                                            /* OF PASSED BUFFER     */
/************************************************************/
/*       WORKING VARIABLES                              @47*/
/************************************************************/
  DCL
   REMBYTES          FIXED BIN(15) INIT(0), /* NUM BYTES TO BE   @47*/
                                            /* PROCESSED IN RTRN AREA*/
   CMDLEN            FIXED BIN(15) INIT(0), /* NUM BYTES IN A    @47*/
                                            /* RETURNED CMD STRING  */
                                            /* RETURN AREA          */
   01 FIXED_BUFF     BASED(ADDR(RETURN_BUFF)),
```

```
      02 FIXED_LEN   FIXED BIN(15),
      02 FIXED_TEXT  CHAR(4160),
                               /* OVERLAY OF PASSED BUF FOR @47*/
                               /* MOVING DATA FROM RETURN AREA */
   FILLBYTS          FIXED BIN(15)  /* NUMBER OF FILL BYTES NEED @47*/
                       INIT(0),     /* TO MAKE RECORD LENGTHS IN    */
                                    /* OUTPUT EQUAL TO BUFROWLN     */
   NUMFULL           FIXED BIN(15)  /* NUMBER OF FULL LINES IN   @47*/
                       INIT(0),     /* PASSED AREA                  */
   PARTROW           FIXED BIN(15)  /* LENGTH OF NON-FULL LINE   @47*/
                       INIT(0),
   J                 FIXED BIN(15)  /* LOOP COUNTER              @47*/
                       INIT(0),
   BUFPOSI           FIXED BIN(15)  /* POSITION IN RETURN BUFFER @47*/
                       INIT(0),
   BUFPOSO           FIXED BIN(15)  /* POSITION IN PASSED BUFFER @47*/
                       INIT(0),
   LEN_CHAR          CHAR(2)        /* LENGTH BYTES IN COMMAND   @47*/
                       INIT(' '),   /* RESULT STRING                */
   LEN_BIT           BIT(16)        /* LENGTH IN BITS FOR        @47*/
                       INIT('0'B),  /* CONVERSION                   */
   LEN_BIN           FIXED BIN(15)  /* LENGTH IN BINARY          @47*/
                       INIT(0),
   SPACE_LEFT        FIXED BIN(15)  /* BYTES LEFT IN BUFFER      @47*/
                       INIT(0);
/************************************************************/
/*       CONSTANTS                                      @47*/
/************************************************************/
 DCL
   BLANK       CHAR(1) INIT(' ') STATIC,      /* BUFFER PADDING  @47*/
   BUFROWLN    FIXED BIN(15) INIT(80) STATIC; /* LNGTH OF A LINE @47*/
                                       /* PASSED TO INVOKER     */
/************************************************************/
/*       DECLARE IFI CALL MACRO DSNWLI                      */
/************************************************************/

  DCL
   DSNWLI         ENTRY EXTERNAL OPTIONS(ASM INTER RETCODE);
                          /* ENTRY POINT IN LANGUAGE INTERFACE    */
                          /* MODULES TO HANDLE IFC API CALLS.     */
%PAGE;
/*****************************************************/
/*  IFCA - (INSTRUMENTATION FACILITY COMMUNICATION  */
/*         AREA) CONTAINS INFORMATION REGARDING THE */
/*         SUCCESS OF THE CALL AND PROVIDES FEEDBACK*/
/*         INFORMATION TO THE APPLICATION PROGRAM.  */
/*                                                  */
/* WARNING:  THIS AREA MUST BE MAINTAINED TO INCLUDE*/
/*           ANY CHANGES TO THE MAPPING MACRO       */
/*           DSNDIFCA                               */
/*                                                  */
/*****************************************************/


  DCL 01 IFCA,
         02 LNGTH              /* LENGTH OF IFCA, INCL LENGTH FIELD*/
              FIXED BIN(15) INIT(0),
         02 UNUSED
              FIXED BIN(15) INIT(0),
         02 EYE_CATCHER        /* USED TO VERIFY THE IFCA BLOCK.   */
              CHAR(4)       INIT( 'IFCA' ),
         02 OWNER_ID           /* TO ESTAB OWNERSHIP OF AN OPN DEST*/
              CHAR(4)       INIT(' '),
         02 IFCARC1            /* RETURN CODE FOR THE IFC API CALL.*/
              FIXED BIN(31) INIT(0),
         02 IFCARC2            /* REASON CODE FOR THE IFC API CALL.*/
              FIXED BIN(31) INIT(0),
         02 BYTES_MOVED        /* BYTES OF RECORD WHICH WERE MOVED.*/
              FIXED BIN(31) INIT(0),
         02 EXCESS_RECDS       /* BYTES OF RECORD WHICH DID NOT FIT*/
              FIXED BIN(31) INIT(0),
         02 OPN_WRIT_SEQ_NUM   /* LAST OPN WRTR SEQ# FOR READA FUNC*/
              FIXED BIN(31) INIT(0),
         02 NUM_RECDS_LOST     /* RECORDS LOST INDICATOR.          */
              FIXED BIN(31) INIT(0),
         02 OPN_NAME_FOR_READA  /* OPN NAME USED FOR READA REQUEST  */
              CHAR(4)       INIT(' '),
         02 OPN_NAMES_AREA,     /* AREA CONTAINING UP TO 8 OPN NAMES*/
            03 OPN_LNGTH        /* LENGTH OF OPN NAMES RETURNED + 4.*/
                 FIXED BIN(15) INIT(0),
            03 UNUSED_2
                 FIXED BIN(15) INIT(0),
```

```
            03 ARRAY_OPN_NAMES(8) /* AREA FOR OPN NAMES RETURNED    */
                 CHAR(4)       INIT(' '),
         02 TRACE_NOS_AREA,      /* AREA CONTAINING UP TO 8 TRACE #'S*/
            03 TRACE_LNGTH       /* LENGTH OF TRACE NO.S RETURNED + 4*/
                 FIXED BIN(15) INIT(0),
            03 UNUSED_3
                 FIXED BIN(15) INIT(0),
            03 ARRAY_TRACE_NOS(8)/* AREA FOR TRACE NUMBERS RETURNED */
                 CHAR(2)       INIT(' '),
         02 DIAGNOS_AREA,        /* DIAGNOSTIC AREA.                 */
            03 DIAGNOS_LNGTH     /* DIAGNOSTIC LENGTH.               */
                 FIXED BIN(15) INIT(0),
            03 UNUSED_4
                 FIXED BIN(15) INIT(0),
            03 DIAGNOS_DATA      /* DIAGNOSTIC DATA.                 */
                 CHAR(80) INIT(' ');

  DCL 01 OUTPUT_AREA,
         02 LNGTH                /* LENGTH OF APPL PGM REC TO WRITE  */
              FIXED BIN(15) INIT(0),
         02 UNUSED
              FIXED BIN(15) INIT(0),
         02 TEXT_OR_COMMAND      /* ACTUAL COMMAND OR RECORD TEXT.   */
              CHAR(254) INIT(' ');

  DCL 01 RETURN_AREA    CTL,     /* COMMAND RESULT AREA              */
         02 LNGTH                /*   NUMBER OF BYTES                */
            FIXED BIN(31),
         02 RTRN_BUFF            /*   OUTPUT BUFFER                  */
            CHAR(*);

/*************************************************************/
/* GENERAL INITIALIZATION                                    */
/*************************************************************/

  FUNCTION   = 'COMMAND';           /* SET FUNCTION FOR IFI CALL     */
  IFCA.LNGTH = STORAGE(IFCA);       /* BYTES USED IN MEMORY          */
  IFCA.EYE_CATCHER = 'IFCA';        /* EYE CATCHER                   */
  IFCA.OWNER_ID = 'LOC2';           /* DB2 LOCATION 1=LOCAL, 2=REMOTE*/
  FREE RETURN_AREA;                 /* FREE STORAGE AND THEN         */
                                    /* ALLOCATE STORAGE FOR THE      */
  ALLOCATE 1 RETURN_AREA,           /* RETURN AREA                   */
             2 LNGTH,
             2 RTRN_BUFF CHAR(4096);

  RTRN_BUFF = ' ';                      /* CLEAR THE RETURN BUFFER   */
  RETURN_AREA.LNGTH = 4096;             /* LENGTH OF RETURN BUFFER   */
  TEXT_OR_COMMAND=BLANK;                /* CLEAR THE DB2 COMMAND AREA*/
  OUTPUT_AREA.UNUSED = '00000000'B;         /* CLEAR THE UNUSED AREA */
  OUTPUT_AREA.LNGTH = LENGTH(INPUTCMD)+4;   /* GET REAL LENGTH OF    */
  OUTPUT_AREA.TEXT_OR_COMMAND = INPUTCMD;   /* ACTUAL DB2 COMMAND    */

  /******************************************/
  /* MAKE THE IFI CALL VIA THE DSNWLI MACRO */
  /******************************************/

  CALL DSNWLI (FUNCTION, IFCA, RETURN_AREA, OUTPUT_AREA);

/********************************************************************/
/* COPY SELECTED VARIABLES FROM IFI COMMAND RESULTS TO OUTPUT       */
/* PARAMETER VARIABLES TO PASS TO REQUESTER PROGRAM FOR PROCESSING. */
/********************************************************************/

  IFCA_RET_HEX = IFCA.IFCARC1;              /* RETURN CODE IN BINARY */
  IFCA_RES_HEX = IFCA.IFCARC2;              /* REASON CODE IN BINARY */
  BUFF_OVERFLOW = 0;             /* PLENTY OF ROOM IN BUFF SO FAR @47*/
  BUFPOSI = 1;                   /* INIT POSITION IN RETURN AREA  @47*/
  BUFPOSO = 1;                   /* INIT POSITION IN PASSED BUFF  @47*/
  /******************************************************************/
  /* COPY RECORDS FROM THE RETURN AREA TO THE CALLER'S BUFFER.   @47*/
  /* PAD EACH RECORD IN THE CALLER'S BUFFER WITH BLANKS SO ITS   @47*/
  /* LENGTH IS A MULTIPLE OF BUFROWLN.                           @47*/
  /******************************************************************/

  IF IFCA.BYTES_MOVED ^= 0 THEN                              /*@47*/
   DO;                               /* IF ANYTHING TO COPY     @47*/
    DO WHILE (BUFPOSI <= IFCA.BYTES_MOVED - 2);              /*@47*/
                                     /* COPY TEXT TO PASSED BUF @47*/
      LEN_CHAR = (SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,2));  /*@47*/
                                     /* GET LENGTH BYTES        @47*/
      LEN_BIT  = UNSPEC(LEN_CHAR);                           /*@47*/
                                     /* CONVERT TO BIT STRING   @47*/
```

```
      LEN_BIN  = LEN_BIT;                                    /*@47*/ 03180000
                                    /* THEN CONVERT TO BINARY   @47*/ 03190000
                                    /* CALC BYTES LEFT IN PASSED@47*/ 03200000
      LEN_BIN = LEN_BIN - 4;        /* TAKE LENGTH BYTES OFF LEN@47*/ 03210000
      SPACE_LEFT = (LEN_BIN / BUFROWLN) * BUFROWLN;          /*@47*/ 03220000
      IF MOD(LEN_BIN,BUFROWLN) > 0 THEN                      /*@47*/ 03230000
        SPACE_LEFT = SPACE_LEFT + BUFROWLN;                  /*@47*/ 03240000
      IF BUFPOSO + SPACE_LEFT - 1 > RETURN_LEN THEN          /*@47*/ 03250000
        BUFF_OVERFLOW = 1;          /* INDICATE BUFFER IS FULL  @47*/ 03260000
      IF BUFF_OVERFLOW = 1 THEN                              /*@47*/ 03270000
        LEAVE;                      /* CAN'T COPY MORE, GET OUT @47*/ 03280000
      BUFPOSI = BUFPOSI + 4;        /* MOVE PAST LENGTH BYTES   @47*/ 03290000
      IF BUFPOSI + LEN_BIN - 1 > IFCA.BYTES_MOVED THEN       /*@47*/ 03300000
        LEAVE;                      /* AT END OF BUFFER         @47*/ 03310000
      NUMFULL = LEN_BIN / BUFROWLN;  /* NUMBER OF FULL LINES     @47*/ 03320000
      PARTROW = MOD(LEN_BIN,BUFROWLN); /* LENGTH OF PARTIAL LINE @47*/ 03330000
      FILLBYTS = BUFROWLN - PARTROW;  /* NUMBER OF PAD BYTES NEED@47*/ 03340000
      IF NUMFULL > 0 THEN            /* MOVE ALL COMPLETE LINES @47*/ 03350000
        DO J = 1 TO NUMFULL;                                 /*@47*/ 03360000
          SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,BUFROWLN) =   /*@47*/ 03370000
            SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,BUFROWLN);  /*@47*/ 03380000
          BUFPOSO = BUFPOSO + BUFROWLN; /* MOVE PAST STRG IN OUTP@47*/ 03390000
          BUFPOSI = BUFPOSI + BUFROWLN; /* MOVE PAST STRG IN INPT@47*/ 03400000
          REMBYTES = REMBYTES - BUFROWLN; /* CALCULATE BYTES LEFT@47*/ 03410000
        END;                                                 /*@47*/ 03420000
      IF PARTROW > 0 THEN                                    /*@47*/ 03430000
        DO;                            /* MOVE PARTIAL LINE     @47*/ 03440000
          SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,PARTROW) =    /*@47*/ 03450000
            SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,PARTROW);   /*@47*/ 03460000
          BUFPOSI = BUFPOSI + PARTROW;  /* MOVE PAST STR IN INPUT@47*/ 03470000
          BUFPOSO = BUFPOSO + PARTROW - 1; /* MOVE TO END OF     @47*/ 03480000
                                          /* STRING IN OUTPUT   @47*/ 03490000
          SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,1) = BLANK;   /*@47*/ 03500000
                                       /* REPLACE THE NEW LINE  @47*/ 03510000
                                       /* CHARACTER IN THE LAST    */ 03520000
                                       /* POSITION WITH A BLANK    */ 03530000
          BUFPOSO = BUFPOSO + 1;       /* MOVE PAST STRG IN OUTP@47*/ 03540000
          REMBYTES = REMBYTES - PARTROW; /* CALCULATE BYTES LEFT @47*/ 03550000
        END;                                                 /*@47*/ 03560000
      IF PARTROW > 0 THEN              /* FILL UP SPACE WITH BLK@47*/ 03570000
        DO;                                                  /*@47*/ 03580000
          DO J = BUFPOSO TO (BUFPOSO + FILLBYTS - 1);        /*@47*/ 03590000
           SUBSTR(FIXED_BUFF.FIXED_TEXT,J,1) = ' ';          /*@47*/ 03600000
          END;                                               /*@47*/ 03610000
          BUFPOSO = BUFPOSO + FILLBYTS; /* MOVE PAST BLANKS     @47*/ 03620000
        END;                                                 /*@47*/ 03630000
    END;                             /* COPY TEXT TO PASSED BUF @47*/ 03640000
    FIXED_BUFF.FIXED_LEN = BUFPOSO - 1;                      /*@47*/ 03650000
                                     /* GET BYTES IN PASSED BUF @47*/ 03660000
  END;                               /* IF ANYTHING TO COPY     @47*/ 03670000
END DSN8EP2;                         /* END PROGRAM                */ 03680000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EPU

PASS Db2 UTILITY STATEMENTS TO BE EXECUTED BY THE STORED PROCEDURE PROGRAM DSNUTILS.

```
DSN8EPU: PROC OPTIONS(MAIN);                                          00010000
/********************************************************************* 00020000
 *   MODULE NAME = DSN8EPU (SAMPLE PROGRAM)                         *  00030000
 *                                                                  *  00040000
 *   DESCRIPTIVE NAME = STORED PROCEDURE REQUESTER PROGRAM          *  00050000
 *                                                                  *  00060000
 *    LICENSED MATERIALS - PROPERTY OF IBM                          *  00070000
 *    5625-DB2                                                      *  00080001
 *    (C) COPYRIGHT 1992, 2003 IBM CORP.                            *  00090001
 *                                                                  *  00100000
 *    STATUS = VERSION 8                                            *  00110001
 *                                                                  *  00120000
 *   FUNCTION =                                                     *  00130000
 *                                                                  *  00140000
 *      PASS DB2 UTILITY STATEMENTS TO BE EXECUTED BY THE STORED    *  00150000
 *      PROCEDURE PROGRAM DSNUTILS.  GET INPUT FROM 'SYSIN'.        *  00160000
 *      PASS THE STATEMENT AND RECEIVE THE OUTPUT RESULTS           *  00170000
 *      VIA A RETURNED CURSOR.  WRITE THE RESULTS TO 'SYSPRINT'.    *  00180000
```

```
 *                                                                  *
 *      DEPENDENCIES = NONE                                         *
 *                                                                  *
 *      RESTRICTIONS =                                              *
 *                                                                  *
 *      INPUT =                                                     *
 *                                                                  *
 *         1. INPUT MUST BE OF THE FORM                             *
 *                                                                  *
 *              Uid='',Restart='',Utility='REORG TABLESPACE',       *
 *              CopyDSN1='SYSADM.COPYDDN.DSN8D51A.DSN8S51E',        *
 *              CopyDEVT1='SYSDA',CopySpace1=10,                    *
 *              Utstmt=                                             *
 *              ' REORG TABLESPACE DSN8D51A.DSN8S51E                *
 *                   SORTDEVT SYSDA SORTNUM 2                       *
 *                   SHRLEVEL CHANGE                                *
 *                   DEADLINE 2010-2-4-03.15.00                     *
 *                   MAPPINGTABLE DSN8510.MAP_TBL                   *
 *                   MAXRO 240 LONGLOG DRAIN DELAY 900              *
 *              ';                                                  *
 *                                                                  *
 *   MODULE TYPE  = PROCEDURE                                       *
 *      PROCESSOR =                                                 *
 *         ADMF PRECOMPILER                                         *
 *         PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE)                 *
 *      MODULE SIZE = 2K                                            *
 *      ATTRIBUTES = RE-ENTERABLE                                   *
 *                                                                  *
 *   ENTRY POINT = DSN8EPU                                          *
 *      PURPOSE  = SEE FUNCTION                                     *
 *      LINKAGE  = STANDARD MVS PROGRAM INVOCATION.                 *
 *      INPUT    = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:   *
 *         SYMBOLIC LABEL/NAME = SYSIN                              *
 *         DESCRIPTION = DDNAME OF SEQUENTIAL DATA SET CONTAINING   *
 *                       DSNUTILS STORED PROCEDURE PARAMETERs.      *
 *      OUTPUT   = PARAMETERS EXPLICITLY RETURNED:                  *
 *         SYMBOLIC LABEL/NAME = SYSPRINT                           *
 *         DESCRIPTION = DDNAME OF SEQUENTIAL OUTPUT DATA SET TO    *
 *                       CONTAIN RESULTS OF THE UTILITIES EXECUTED. *
 *                                                                  *
 *   EXIT NORMAL =                                                  *
 *                                                                  *
 *   NORMAL MESSAGES =                                              *
 *                                                                  *
 *   EXIT-ERROR =                                                   *
 *                                                                  *
 *                                                                  *
 *      ABEND CODES = NONE                                          *
 *                                                                  *
 *   ERROR MESSAGES =                                               *
 *                                                                  *
 *   EXTERNAL REFERENCES =                                          *
 *      ROUTINES/SERVICES = NONE                                    *
 *      DATA-AREAS        = NONE                                    *
 *      CONTROL-BLOCKS =                                            *
 *         SQLCA               - SQL COMMUNICATION AREA             *
 *                                                                  *
 *   PSEUDOCODE      =                                              *
 *                                                                  *
 *   DSN8EPU: PROCEDURE.                                            *
 *   DECLARATIONS.                                                  *
 *   INITIALIZE VARIABLES.                                          *
 *   GET THE INPUT PARAMETERS AND COPY TO SYSPRINT.                 *
 *   EXEC SQL CALL SYSPROC.DSNUTILS.                                *
 *   DO UNTIL SQLCODE > 0.                                          *
 *      EXEC SQL FETCH FROM RESULT SET.                             *
 *      PRINT RESULT SET TO SYSPRINT.                               *
 *   END.                                                           *
 *                                                                  *
 *  NOTICE =                                                        *
 *    THIS SAMPLE PROGRAM USES DB2 UTILITIES. SOME UTILITY FUNCTIONS*
 *    ARE ELEMENTS OF SEPARATELY ORDERABLE PRODUCTS.  SUCCESSFUL USE*
 *    OF A PARTICULAR SAMPLE MAY BE DEPENDENT UPON THE OPTIONAL     *
 *    PRODUCT BEING LICENSED AND INSTALLED IN YOUR ENVIRONMENT.     *
 *                                                                  *
 *  CHANGE ACTIVITY =                                               *
 *   PQ24720 - Add FILTRDSN and Fix I/O for seq #ed input           *
 *   PQ44916 - Fix code hole closed by VA and Enterprise PL/I       *
 *   d54292  - Check for unexpected SQLCODE in FETCH loop           *
 *******************************************************************/
DCL SYSPRINT           FILE OUTPUT STREAM;
```

```
DCL SYSIN             FILE INPUT STREAM   ENV( F RECSIZE(80) );

DCL 01 SYSIN_REC,
       05 UTIL_OPTS   CHAR( 72 ),
       05 SEQ_NOS     CHAR( 08 );

DCL SYSIN_EOF         BIT( 01 )           INIT( '0'B );
ON ENDFILE( SYSIN )
 SYSIN_EOF = '1'B;

DCL UTIL_OPTS_BUFF    VARYING CHAR( 32760 ) INIT( '' );

DCL ADDR              BUILTIN;
DCL NULL              BUILTIN;
DCL PLIRETC           BUILTIN;

DCL UID      CHAR(16) VARYING; /* UTILITY ID    */
DCL RESTART  CHAR(8) VARYING;  /* RESTART         */
DCL UTSTMT     CHAR(32704) VARYING;
DCL RETCODE    FIXED BIN(31);
DCL UTILITY  CHAR(20) VARYING;
DCL RECDSN     CHAR(44) VARYING,
    RECDEVT    CHAR(8),
    RECSPACE   FIXED BIN(15);
DCL DISCDSN    CHAR(44) VARYING,
    DISCDEVT   CHAR(8),
    DISCSPACE  FIXED BIN(15);
DCL PNCHDSN    CHAR(44) VARYING,
    PNCHDEVT   CHAR(8),
    PNCHSPACE  FIXED BIN(15);
DCL COPYDSN1   CHAR(44) VARYING,
    COPYDEVT1  CHAR(8),
    COPYSPACE1 FIXED BIN(15);
DCL COPYDSN2   CHAR(44) VARYING,
    COPYDEVT2  CHAR(8),
    COPYSPACE2 FIXED BIN(15);
DCL RCPYDSN1   CHAR(44) VARYING,
    RCPYDEVT1  CHAR(8),
    RCPYSPACE1 FIXED BIN(15);
DCL RCPYDSN2   CHAR(44) VARYING,
    RCPYDEVT2  CHAR(8),
    RCPYSPACE2 FIXED BIN(15);
DCL WORKDSN1   CHAR(44) VARYING,
    WORKDEVT1  CHAR(8),
    WORKSPACE1 FIXED BIN(15);
DCL WORKDSN2   CHAR(44) VARYING,
    WORKDEVT2  CHAR(8),
    WORKSPACE2 FIXED BIN(15);
DCL MAPDSN     CHAR(44) VARYING,
    MAPDEVT    CHAR(8),
    MAPSPACE   FIXED BIN(15);
DCL ERRDSN     CHAR(44) VARYING,
    ERRDEVT    CHAR(8),
    ERRSPACE   FIXED BIN(15);
DCL FILTRDSN   CHAR(44) VARYING,
    FILTRDEVT  CHAR(8),
    FILTRSPACE FIXED BIN(15);
DCL RESULTS  SQL TYPE IS RESULT_SET_LOCATOR VARYING;
DCL SEQNO FIXED BIN(31);
DCL TEXT CHAR(122) VARYING;
EXEC SQL INCLUDE SQLCA;
Uid='';
Restart='';
Utstmt='';
RetCode = 0;
Utility='';
RecDSN='';   RecDEVT='';   RecSpace=0;
DiscDSN='';  DiscDEVT='';  DiscSpace=0;
PnchDSN='';  PnchDEVT='';  PnchSpace=0;
CopyDSN1=''; CopyDEVT1=''; CopySpace1=0;
CopyDSN2=''; CopyDEVT2=''; CopySpace2=0;
RcpyDSN1=''; RcpyDEVT1=''; RcpySpace1=0;
RcpyDSN2=''; RcpyDEVT2=''; RcpySpace2=0;
WorkDSN1=''; WorkDEVT1=''; WorkSpace1=0;
WorkDSN2=''; WorkDEVT2=''; WorkSpace2=0;
MapDSN='';   MapDEVT='';   MapSpace=0;
ErrDSN='';   ErrDEVT='';   ErrSpace=0;
FiltrDSN=''; FiltrDEVT=''; FiltrSPACE=0;

/* Collect DSNUTILS options from SYSIN records, columns 1-72        */
GET COPY EDIT( UTIL_OPTS,SEQ_NOS ) ( A(72),A(8) );
DO WHILE( ^SYSIN_EOF );
```

```
  UTIL_OPTS_BUFF = UTIL_OPTS_BUFF || UTIL_OPTS;
  GET COPY EDIT( UTIL_OPTS,SEQ_NOS )( A(72),A(8) );
END; /* DO WHILE( ^SYSIN_EOF ); */

/* Assign DSNUTILS options from inputted settings in UTIL_OPTS_BUFF */
GET STRING( UTIL_OPTS_BUFF ) DATA;

/* Call DSNUTILS stored procedure to process the inputted settings  */
EXEC SQL
 CALL SYSPROC.DSNUTILS(:UID, :RESTART,
               :UTSTMT,
               :RETCODE,
               :UTILITY,
               :RECDSN  ,:RECDEVT  ,:RECSPACE  ,
               :DISCDSN ,:DISCDEVT ,:DISCSPACE ,
               :PNCHDSN ,:PNCHDEVT ,:PNCHSPACE ,
               :COPYDSN1,:COPYDEVT1,:COPYSPACE1,
               :COPYDSN2,:COPYDEVT2,:COPYSPACE2,
               :RCPYDSN1,:RCPYDEVT1,:RCPYSPACE1,
               :RCPYDSN2,:RCPYDEVT2,:RCPYSPACE2,
               :WORKDSN1,:WORKDEVT1,:WORKSPACE1,
               :WORKDSN2,:WORKDEVT2,:WORKSPACE2,
               :MAPDSN  ,:MAPDEVT  ,:MAPSPACE  ,
               :ERRDSN  ,:ERRDEVT  ,:ERRSPACE  ,
               :FILTRDSN,:FILTRDEVT,:FILTRSPACE);
IF SQLCODE < 0 THEN
  DO;
    PUT SKIP EDIT('CALL SQLCA')(A);
    PUT SKIP DATA(SQLCA);
    CALL PLIRETC(8);            /* SET PLI RETURN CODE            */
    RETURN;
  END;

EXEC SQL
 ASSOCIATE LOCATOR (:RESULTS) WITH PROCEDURE SYSPROC.DSNUTILS;
IF SQLCODE < 0 THEN
  DO;
    PUT SKIP EDIT('ASSOCIATE LOCATOR SQLCA')(A);
    PUT SKIP DATA(SQLCA);
    CALL PLIRETC(8);            /* SET PLI RETURN CODE            */
    RETURN;
  END;

EXEC SQL
 ALLOCATE SYSPRINT CURSOR FOR RESULT SET :RESULTS;
IF SQLCODE < 0 THEN
  DO;
    PUT SKIP EDIT('ALLOCATE SYSPRINT SQLCA')(A);
    PUT SKIP DATA(SQLCA);
    CALL PLIRETC(8);            /* SET PLI RETURN CODE            */
    RETURN;
  END;

FETCHLOOP:
DO UNTIL(SQLCODE < 0 | SQLCODE = 100 );
  EXEC SQL
   FETCH SYSPRINT INTO :SEQNO, :TEXT;
  IF (SQLCODE >= 0)
   & (SQLCODE ^= 100) THEN
    DO;
      PUT SKIP EDIT(TEXT)(A);
    END;
  IF (SQLCODE ^= 0)
   & (SQLCODE ^= 100) THEN
    DO;
      PUT SKIP EDIT('FETCH SYSPRINT SQLCA')(A);
      PUT SKIP DATA(SQLCA);
    END;
END FETCHLOOP;
IF SQLCODE < 0 THEN
  DO;
    CALL PLIRETC(8);            /* SET PLI RETURN CODE            */
    RETURN;
  END;

PUT SKIP DATA(RetCode);
CALL PLIRETC(RetCode);          /* SET PLI RETURN CODE            */
END DSN8EPU;
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ED1

Pass Db2 commands received from standard input to stored procedure DSN8ED2 for execution.

```
/*********************************************************************/
/*   Module name = DSN8ED1 (sample program)                          */
/*                                                                   */
/*   DESCRIPTIVE NAME = Stored procedure result set requester pgm    */
/*                                                                   */
/*    LICENSED MATERIALS - PROPERTY OF IBM                           */
/*    5645-DB2                                                       */
/*    (C) COPYRIGHT 1998 IBM CORP.  ALL RIGHTS RESERVED.             */
/*                                                                   */
/*    STATUS = VERSION 6                                             */
/*                                                                   */
/*   Function =                                                      */
/*                                                                   */
/*      Pass DB2 commands received from standard input to            */
/*      stored procedure DSN8ED2 for execution.  Receive the         */
/*      command results from DSN8ED2 as result set.  Unload the      */
/*      result set and print the contents to standard output.        */
/*                                                                   */
/*      Dependencies = None                                          */
/*                                                                   */
/*      Restrictions =                                               */
/*                                                                   */
/*         1. DB2 commands must be preceded by a hyphen and          */
/*            followed by a semicolon. Lines with an asterisk        */
/*            in the first column or two hyphens as the first        */
/*            nonblank characters are interpreted as comment         */
/*            lines.  Two hyphens placed after the command text in   */
/*            in a line indicate that the rest of the line is        */
/*            comments only.                                         */
/*                                                                   */
/*         2. A command may be no more than 4096 bytes.              */
/*                                                                   */
/*      Input =                                                      */
/*         1. A single input parameter that indicates the location   */
/*            of the stored procedure. The contents must be a        */
/*            valid DB2 location name of at most 16 characters.      */
/*                                                                   */
/*         2. Lines of length INPUTL from standard input.            */
/*            Only the first INPUSED bytes are used. Lines are       */
/*            considered to be either command text or comments.      */
/*            Command text begins with a hyphen and ends with a      */
/*            semicolon. Comments begin with an asterisk in          */
/*            column one or two hyphens as the first two nonblank    */
/*            characters. Command text may span lines, but comment   */
/*            text may not.                                          */
/*                                                                   */
/*      Output =                                                     */
/*            Lines of length OUTLEN to standard output.             */
/*            Each line contains one of the following:               */
/*            a. Command text.                                       */
/*            b. Results returned by the DB2 for MVS/ESA command     */
/*               processor after the command is issued.              */
/*                                                                   */
/*   Module type  = C program                                        */
/*      Processor =                                                  */
/*         ADMF Precompiler                                          */
/*         C/370                                                     */
/*      Module size = See linkedit output                            */
/*      Attributes = Not reentrant or reusable                       */
/*                                                                   */
/*   Entry point = DSN8ED1                                           */
/*      Purpose  = See Function                                      */
/*      Linkage  = Standard MVS program invocation, one parameter.   */
/*                                                                   */
/*   Exit normal =                                                   */
/*                                                                   */
/*      Return code 0 on normal completion.                          */
/*                                                                   */
/*   Normal messages =                                               */
/*                                                                   */
```

```
/*      *** Input statement:  <DB2 command input statement text>    */
/*      *** IFI return area:  <Results of DB2 command execution>    */
/*                                                                  */
/*                                                                  */
/*   Exit-error =                                                   */
/*                                                                  */
/*      Return code =  4 - Warnings occurred.                       */
/*        -  The DB2 for MVS/ESA Instrumentation Facility Interface */
/*           (IFI) invocation of the DB2 command resulted in a      */
/*           return code 4. The accompanying reason code indicates  */
/*           the specific problem.                                  */
/*                                                                  */
/*      Return code =  8 - Errors occurred.                         */
/*        -  The DB2 for MVS/ESA Instrumentation Facility Interface */
/*           (IFI) invocation of the DB2 command resulted in a      */
/*           return code 8. The accompanying reason code indicates  */
/*           the specific problem.                                  */
/*                                                                  */
/*      Return code = 12 - Severe errors occurred.                  */
/*        -  Input parameter 1 did not contain the name of the      */
/*           DB2 server where the stored procedure resides.         */
/*        -  The input dataset (SYSIN) did not contain any data.    */
/*        -  Command input did not begin with a hyphen.             */
/*        -  Command input was not ended with a semicolon.          */
/*        -  An input statement contained more than STMTMAX        */
/*           bytes.                                                 */
/*        -  Connection to the stored procedure location failed.    */
/*        -  The SQL CALL statement to the stored procedure failed. */
/*        -  The DB2 for MVS/ESA Instrumentation Facility Interface */
/*           (IFI) invocation of the DB2 command resulted in a      */
/*           return code 12. The accompanying reason code indicates */
/*           the specific problem.                                  */
/*        -  The call to the stored procedure, DSN8ED2, succeeded   */
/*           but DSN8ED2 experienced SQL problems.  The formatted   */
/*           SQL error message appears in SYSPRINT.                 */
/*        -  The call to the stored procedure, DSN8ED2, succeeded   */
/*           but no result set was returned.  SYSPRINT messages     */
/*           should provide more information.                       */
/*        -  A result set was returned by DSN8ED2 but one of the    */
/*           following occurred (see SYSPRINT messages for details):*/
/*           - The locator variable could not be associated with    */
/*             the result set.                                      */
/*           - The result set cursor could not be allocated         */
/*           - No data could be fetched from the result set cursor. */
/*                                                                  */
/*      Abend codes = None                                          */
/*                                                                  */
/*   Error messages =                                               */
/*      -  *** ERROR: No server name provided - DSN8ED1 ended.      */
/*      -  *** ERROR: No input records found - DSN8ED1 ended.       */
/*      -  *** ERROR: Syntax for DB2 command is invalid.            */
/*         ***        A valid command ends with a semicolon.        */
/*      -  *** ERROR: Syntax for DB2 command is invalid.            */
/*         ***        A valid command begins with a hyphen.         */
/*      -  *** ERROR: Statement length is greater than the ____     */
/*                    character maximum.                            */
/*      -  *** ERROR: Connection to server <location> was unsuc-    */
/*                    cessful.                                      */
/*      -  *** ERROR: Call to stored procedure DSN8ED2 failed;      */
/*                    diagnostics follow.                           */
/*      -  *** ERROR: The following diagnostics were returned by    */
/*                    stored procedure DSN8ED2.                     */
/*      -  *** ERROR: DSNTIAR could not format the message.         */
/*                    SQLCODE is ____, SQLERRM is ___________ ...   */
/*      -  *** WARNING: Call to stored procedure DSN8ED2 succeeded  */
/*                      but no result set was returned.             */
/*      -  *** WARNING: IFI error codes returned by DSN8ED2.        */
/*                      Return code=___, reason code=____ from      */
/*                      IFI request.                                */
/*      -  *** WARNING: ____ records were lost because the IFI      */
/*                      return area in stored procedure DSN8ED2     */
/*                      is too small to accomodate this request.    */
/*                   ** Increase the IFI return area (RETURN_LEN)   */
/*                      in DSN8ED2 and then recompile/relink/rebind */
/*                      before resubmitting this request.           */
/*                                                                  */
/*      -  *** Syntax for DB2 command is invalid.                   */
/*         *** A valid command must begin with a hyphen.            */
/*                                                                  */
/*      -  *** Statement length is greater than the <STMTMAX>       */
/*             character maximum.                                   */
/*      -  *** Connection to <location> unsuccessful.               */
```

```
/*         *** SQLCODE is <sqlcode>.                              */
/*      -  *** Call to DSN8ED2 unsuccessful.                      */
/*         *** SQLCODE is <sqlcode>.                              */
/*      -  *** Insufficient space to receive all output from IFI    */
/*             return area.                                        */
/*      -  *** Return code=<return code>, reason code=<reason code> */
/*             from IFI request.                                   */
/*      -  *** Severe error occurred. Program is terminating.      */
/*                                                                 */
/*   External references =                                         */
/*      Routines/services =                                        */
/*         none                                                    */
/*      Data areas =                                               */
/*         none                                                    */
/*      Control blocks =                                           */
/*         SQLCA - SQL communication area                          */
/*                                                                 */
/*   Pseudocode =                                                  */
/*                                                                 */
/*   DSN8ED1:                                                      */
/*   - Extract the name of the DB2 server where stored procedure    */
/*     DSN8ED2 resides from input parameter number 1.              */
/*   - Call build_DB2_command to create a logical DB2 command record*/
/*     from one or more records from SYSIN.                        */
/*   - If a command was created successfully, do the following until*/
/*     all input has been processed or severe errors occur.        */
/*       - Call connect_to_sp_server to connect to the DB2 server   */
/*         specified in the first input parameter.                 */
/*       - Call send_DB2_command_to_sp to invoke stored procedure   */
/*         DSN8ED2 to process the command.                         */
/*       - Call output_results_from_sp to unload the result set     */
/*         from DSN8ED2 to SYSPRINT.                               */
/*       - Call build_DB2_command to create the next logical DB2    */
/*         command record from SYSIN records.                      */
/*   End DSN8ED1                                                   */
/*                                                                 */
/*   build_DB2_command:                                            */
/*   - Read a record from SYSIN                                    */
/*   - Do the following until either a full command is built, end   */
/*     of file is reached, or an error occurs:                     */
/*     - if the first byte of the record is '*' or the first two    */
/*       nonblank bytes of the record are '--' then the whole       */
/*       record is a comment.  Disregard it.                       */
/*     - if '--' is encountered after nonblanks are found then the  */
/*       rest of the record is a comment and can be disregarded.    */
/*     - else if a semicolon is found inside a delimited string     */
/*       call copy_byte_to_cmd_buf to add it to the command string. */
/*       - a delimited string is one that starts but has not yet    */
/*         terminated with a single quote or a double quote         */
/*     - else if a semicolon is found outside a delimited string    */
/*       then the command is complete.                             */
/*     - else if a nonblank is found then call copy_byte_to_cmd_buf */
/*       to add it to the command string.                          */
/*     - else if a blank is found inside a delimited string then    */
/*       call copy_byte_to_cmd_buf to add it to the command string. */
/*     - else if a blank is found outside a delimited string and    */
/*       the preceding byte was nonblank then call copy_byte_to_    */
/*       cmd_buf to add it to the command string.                  */
/*     - else if a blank is found outside a delimited string and    */
/*       the preceding byte was blank then disregard the blank.     */
/*     - if the input record is exhausted before a terminating      */
/*       semicolon is found, read the next input record.           */
/*   - When a command is created successfully, call echo_DB2_command*/
/*     to output the reformatted command.                          */
/*   - Check the command to ensure that it starts with a hypen.     */
/*   End build_DB2_command                                         */
/*                                                                 */
/*   copy_byte_to_cmd_buf                                          */
/*   - append the current byte of the input record to the end of    */
/*     the command string and update length of command string.      */
/*   - if command string exceeds buffer size, issue a message and   */
/*     end DSN8ED1.                                                */
/*   End copy_byte_to_cmd_buf                                      */
/*                                                                 */
/*   echo_DB2_command                                              */
/*   - output the reformatted DB2 command to SYSPRINT              */
/*   End echo_DB2_command                                          */
/*                                                                 */
/*   connect_to_sp_server                                          */
/*   - invoke SQL to CONNECT to the DB2 server where stored proc-   */
/*     edure DSN8ED2 resides.                                      */
/*   - if the CONNECT fails, issue a message and end DSN8ED1.      */
```

```
/*   End connect_to_sp_server                                         */
/*                                                                    */
/*   send_DB2_command_to_sp                                           */
/*   - invoke SQL to call stored procedure DSN8ED2 to process the     */
/*     contents of the command buffer.                                */
/*   - analyze the resultant SQLCODE, IFI return and result codes,    */
/*     and buffer overload and error parameters returned by DSN8ED2.*/
/*   End send_DB2_command_to_sp                                       */
/*                                                                    */
/*   output_results_from_sp                                           */
/*   - associate a DB2 locator variable with the result set from      */
/*     stored procedure DSN8ED2                                       */
/*   - allocate a cursor to the result set                            */
/*   - fetch each row from the result set and output it to SYSPRINT */
/*   End output_results_from_sp                                       */
/*                                                                    */
/*   sql_error                                                        */
/*   - invoke DSNTIAR to format the current SQL code and print the    */
/*     messages to SYSPRINT                                           */
/*   - if DSNTIAR cannot detail the code, output the SQLCODE and      */
/*     SQLERRM to SYSPRINT
/*   End sql_error                                                    */
/*                                                                    */
/*  Change activity =                                                 */
/*    none                                                            */
/*********************************************************************/

/********************** C library definitions **********************/
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>

/**************************** Constants ****************************/
#define     INPUTL         81         /* Length of input line       */
#define     INPUSED        72         /* Bytes used in an input line*/
#define     LOCLEN         16         /* Length of input parm (loc) */
#define     OUTLEN         81         /* Length of output line      */
#define     RETWRN          4         /* Warning return code        */
#define     RETERR          8         /* Error return code          */
#define     RETSEV         12         /* Severe error return code   */
#define     STMTMAX      4096         /* Maximum statement length   */
#define     ASTERISK      '*'         /* Comment indicator          */
#define     BLANK         ' '         /* Blank                      */
#define     HYPHEN        '-'         /* Hyphen                     */
#define     NULLCHAR     '\0'         /* Null character             */
#define     QUOTE        '\''         /* Quotation mark             */
#define     DQUOTE        '"'         /* Double quote               */
#define     SEMICOLON     ';'         /* SQL stmt terminator        */

enum flag   {No, Yes};                /* Settings for flags         */

/********************** Program Argument List **********************/
char *parms[];                        /* Contains input parameter   */

/********************** Standard Input/Output **********************/
FILE        *sysin;                   /* Input statements           */
char        input[INPUTL];            /* Current input data         */
char        *inres;                   /* Result of gets invocation  */

FILE        *sysprint;                /* Command results/error msgs */

/************************ Working variables ************************/
short int   c;                        /* pointer to command buffer  */
enum flag   dquotflag;                /* '"' delimiter status       */
short int   i;                        /* pointer to input buffer    */
short int   j;                        /* miscellaneous counter, ptr */
enum flag   endstr;                   /* End of statement flag      */
enum flag   input_eof;                /* End of input data flag     */
enum flag   quoteflag;                /* "'" delimiter status       */

/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
char        db2loc2[17];              /* Remote DB2 location name   */
long int    ifca_ret_hex;             /* Return code from IFI call  */
long int    ifca_res_hex;             /* Reason code from IFI call  */
long int    xs_bytes_hex;             /* No. of bytes not returned  */
long int    rc;                       /* All-purpose return var     */
struct {
  short int   sp_err_blen;            /* Error msg buffer length    */
  char        sp_err_txt[880];        /* Error msg text             */
     }    sp_err_buf;                 /* Error message buffer from  */
                                      /*  stored procedure          */
```

```
short int   sperind1;                  /* Indicator vars for parm 1  */
short int   sperind2;                  /* Indicator vars for parm 2  */
short int   sperind3;                  /* Indicator vars for parm 3  */
short int   sperind4;                  /* Indicator vars for parm 4  */
short int   sperind5;                  /* Indicator vars for parm 5  */

struct {
  short int   cmdlen;                  /* Statement length           */
  char        cmdtxt[4096];            /* Statement text             */
     }    cmdbuf;                      /* Statement buffer passed to */
                                       /*  stored procedure          */

                                       /* Result set locator         */
static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ED2_rs_loc;

long int    rs_sequence;               /* Result set table data sequ */
char        rs_data[80];               /* Result set data buffer     */
                                       /* - length is OUTLEN - 1     */
EXEC SQL END DECLARE SECTION;

/******************* DB2 SQL Communication Area *******************/
EXEC SQL INCLUDE SQLCA;


/*****************************************************************
******************************************************************
** main routine                                                 **
******************************************************************
*****************************************************************/
int main( int argc, char *argv[] )                          /*proc*/
{
  /***************************************************************
  * initialize working variables                                 *
  ***************************************************************/
  cmdbuf.cmdlen = 0;                   /* Nothing in command buf yet */
  input_eof = No;                      /* Not at end of input        */
  rc = 0;                              /* No errors yet              */

  sperind1 = -1;                       /* Clear null indicator var 1 */
  sperind2 = -1;                       /* Clear null indicator var 2 */
  sperind3 = -1;                       /* Clear null indicator var 3 */
  sperind4 = -1;                       /* Clear null indicator var 4 */
  sperind5 = -1;                       /* Clear null indicator var 5 */

  /***************************************************************
  * get input parameter (name of server where stored proc resides)   *
  ***************************************************************/
  for( j=1; j<argc; j++ )              /* break out the input parms  */
    parms[j] = argv[j];

  for( j=0; j<LOCLEN; j++ )            /* Extract name of DB2 server */
    db2loc2[j] = *(parms[1]+j);        /*  where sp resides          */

  if( db2loc2[1] == BLANK )            /* If no server specified,    */
  {                                    /*  issue error               */
    printf( " *** ERROR: No server name provided - DSN8ED1 ended.\n" );
    rc = RETSEV;
  }

  db2loc2[j]=NULLCHAR;                 /* Null-terminate the string  */

  /***************************************************************
  * build the first DB2 command from one or more input records      *
  ***************************************************************/
  if( rc < RETSEV )
  {
    build_DB2_command();
    if( input_eof == Yes  &&  rc < RETSEV )
    {
      printf( " *** ERROR: No input records found - DSN8ED1 ended.\n");
      rc = RETSEV;
    }
  }

  /***************************************************************/
  /* If a command was built successfully, connect to the DB2 server */
  /* where the stored procedure resides, send the command to the    */
  /* stored procedure, output the results, and build the next com-  */
  /* mand, if any.                                                  */
  /***************************************************************/
  while( input_eof == No  &&  rc < RETSEV )
  {
```

```
    connect_to_sp_server();           /* connect to the server      */

    if( rc < RETSEV )                 /* if successful              */
      send_DB2_command_to_sp();       /*  invoke the stored proc    */

    if( rc < RETSEV )                 /* if successful              */
      output_results_from_sp();       /*  out the results           */

    if( rc < RETSEV )                 /* if successful              */
      build_DB2_command();            /*  process the next input    */
  }

  printf( " \n \n *** DSN8ED1 completed; highest return code was %d\n",
          rc );

  return( rc );                       /* put return code in ctl blk */


} /* end of main program */


/*********************************************************************
**********************************************************************
** Build a DB2 command from one or more physical input records     **
**********************************************************************
*********************************************************************/
build_DB2_command()                                          /*proc*/
{
  /*******************************************************************
  * initialize working variables                                     *
  *******************************************************************/
  for( i=0; i<INPUTL; i++ )           /* Blank the input array      */
    input[i] = ' ';

  c = 0;                              /* marks pos'n in command buf */
  cmdbuf.cmdlen = 0;                  /* no. of bytes in cmd buffer */
  dquotflag = No;                     /* flags delimiter stat of '"'*/
  endstr = No;                        /* flags end of log inp record*/
  i = 0;                              /* marks pos'n in input buffer*/
  quoteflag = No;                     /* flags delimiter stat of "'"*/

  /*******************************************************************
  * read the first physical record of the command from input         *
  *******************************************************************/
  inres = gets( input );
  if( inres == NULL )                 /* If end of file reached     */
    input_eof = Yes;                  /*  then all finished         */

  /*******************************************************************
  * parse the current input record for DB2 command parts             *
  *******************************************************************/
  while( endstr == No  &&  input_eof == No  &&  rc < RETSEV )
  {
    /*****************************************************************
    * If 1st char in a line is '*' -OR- the 1st two non-blank chars  *
    * in a line are '--', this is a comment line.  Don't copy it to  *
    * the command buffer; request next line.                         *
    *****************************************************************/
    if( i == 0  &&  input[0] == ASTERISK )
      i = INPUSED;
    else if( cmdbuf.cmdlen == 0
         && input[0] == HYPHEN  &&  input[1] == HYPHEN )
      i = INPUSED;

    /*****************************************************************
    * Otherwise, this must be a command line.  Parse it into the com-*
    * mand buffer while looking for delimiters and the end of stmt.  *
    *****************************************************************/
    else
      while( i < INPUSED  &&  endstr == No  &&  rc < RETSEV )
      {
        if( input[i] == DQUOTE )      /* if double quote found      */
          if( dquotflag == Yes )      /*  and is already a delimiter*/
            dquotflag = No;           /*   note end of delimited str*/
          else if( quoteflag == No )  /*  else if it's not delimited*/
            dquotflag = Yes;          /*   then it's a delimiter    */

        if( input[i] == QUOTE )       /* if single quote found      */
          if( quoteflag == Yes )      /*  and is already a delimiter*/
            quoteflag = No;           /*  -note end of delimited str*/
          else if( dquotflag == No )  /*  else if it's not delimited*/
            quoteflag = Yes;          /*  then it's a delimiter     */
```

```
        if( input[i] == HYPHEN        /* if '--' found in current   */
        &&  input[i+1] == HYPHEN      /*  and next byte             */
         && i < INPUSED               /*  not at end of input line  */
         && quoteflag == No           /*  and not in a delimited    */
         && dquotflag == No )         /*  strng then rest is comment*/
          i = INPUSED;                /*   ignore it; rqst next line*/

        else if( input[i] == SEMICOLON/* else if semicolon found    */
         && quoteflag == No           /*  and it's not delimited    */
         && dquotflag == No )         /*  then command is complete  */
          endstr = Yes;               /*   fall through             */

        else if( input[i] != BLANK )  /* else if non-blank found    */
          copy_byte_to_cmd_buf();     /*  copy it to command buffer */

        else if( input[i] == BLANK    /* else if blank found        */
         && ( quoteflag == Yes        /*  and it's in a delimited   */
           || dquotflag == Yes ) )    /*  string                    */
          copy_byte_to_cmd_buf();     /*   copy it to command buffer*/

        else if( input[i] == BLANK    /* else if blank found        */
         && c > 0                     /*  and something's in cmd buf*/
         && cmdbuf.cmdtxt[c-1]!=BLANK)/* and prev cmd byte nonblank */
          copy_byte_to_cmd_buf();     /*   copy it to command buffer*/

        else;                         /* swallow all other blanks   */

        i++;                          /* bump pos'n in input record */

      } /* end while( i<INPUSED  &&  endstr == No  &&  rc<RETSEV )  */

    /****************************************************************
    * if current physical record is exhausted but the current log-   *
    * ical record is still incomplete, get the next physical record  *
    ****************************************************************/
    if( i >= INPUSED  &&  endstr == No  &&  rc < RETSEV )
    {
      for( i=0; i<INPUTL; i++ )       /* Blank the input array      */
        input[i] = ' ';
      i = 0;                          /* reset pointer to input buff*/
      inres = gets( input );          /* Read the next physical rec */
      if( inres == NULL )             /* If end of file reached     */
      {                               /*  current logical rec inmplt*/
        input_eof = Yes;              /*  don't ask for more        */
        printf( " *** ERROR: Syntax for DB2 command is invalid.\n");
        printf( " ***        A valid command ends with a" );
        printf( " semicolon.\n" );
        rc = RETSEV;                  /*  stop the program          */
      }
    }
  } /* end while( endstr == No && input_eof == No && rc < RETSEV )  */

  /******************************************************************
  * display the reformatted command (if one exists)                  *
  ******************************************************************/
  if( cmdbuf.cmdlen > 0 )
    echo_DB2_command();

  /******************************************************************
  * verify that the command has a valid syntax                       *
  ******************************************************************/
  if( endstr == Yes  &&  input_eof == No  &&  rc < RETSEV )
    if( cmdbuf.cmdtxt[0] != HYPHEN )
    {
      printf( " *** ERROR: Syntax for DB2 command is invalid.\n");
      printf( " ***        A valid command begins with a hyphen.\n" );
      rc = RETSEV;
    }
} /* end of build_DB2_command() */


/*******************************************************************
********************************************************************
** Copy the current byte of current input line to command buffer    **
********************************************************************
*******************************************************************/
copy_byte_to_cmd_buf()                                       /*proc*/
{
  cmdbuf.cmdtxt[c++] = input[i];
  cmdbuf.cmdlen = c;
```

```
  /******************************************************************
  * if entry is too long for command buffer, issue message and quit  *
  ******************************************************************/
  if( cmdbuf.cmdlen >= STMTMAX )
  {
    printf( " *** ERROR: Statement length is greater than the" );
    printf( " %d character maximum.\n",STMTMAX );
    rc = RETSEV;
  }
} /* end of copy_byte_to_cmd_buf() */


/*********************************************************************
*********************************************************************
** Connect to the server where the stored procedure resides        **
*********************************************************************
*********************************************************************/
connect_to_sp_server()                                          /*proc*/
{
  EXEC SQL CONNECT TO :db2loc2;
  if( SQLCODE != 0 )
  {
    printf( " *** ERROR: Connection to server %s was unsuccessful.\n",
            db2loc2 );
    sql_error( " *** Connection to server unsuccessful" );
    rc = RETSEV;
  }
} /* end of connect_to_sp_server() */


/*********************************************************************
*********************************************************************
** Process the current DB2 command built from the input file       **
*********************************************************************
*********************************************************************/
send_DB2_command_to_sp()                                        /*proc*/
{
  sperind1 = 0;                          /* tell DB2 to transmit    */
                                         /*  contents of parm 1     */
  EXEC SQL CALL DSN8.DSN8ED2(       :cmdbuf :sperind1,
                            :ifca_ret_hex :sperind2,
                            :ifca_res_hex :sperind3,
                            :xs_bytes_hex :sperind4,
                              :sp_err_buf :sperind5 );

  /******************************************************************
  * verify the SQL return code returned by the stored procedure      *
  ******************************************************************/
  if( SQLCODE == 0 )
  {
    printf( " *** WARNING: Call to stored procedure DSN8ED2" );
    printf(     " succeeded\n" );
    printf( "              but no result set was returned.\n" );
    if( rc < RETERR )
      rc = RETERR;
  }
  else if( SQLCODE == 466 )
  {
    printf( " *** A result set was returned by stored procedure" );
    printf(     " DSN8ED2.\n" );
  }
  else
  {
    printf( " *** ERROR: Call to stored procedure DSN8ED2 failed;" );
    printf(     " diagnostics follow.\n" );
    sql_error( "*** Stored procedure call unsuccessful." );
    rc = RETSEV;
  }

  /******************************************************************
  * verify the IFI return code returned by the stored procedure      *
  ******************************************************************/
  if( sperind2 != -1  &&  ifca_ret_hex != 0 )
  {
    printf( " *** WARNING: IFI error codes returned by DSN8ED2.\n" );
    printf( " ***          Return code=%0X ", ifca_ret_hex );
    printf( " *** reason code=%0X from IFI request.\n", ifca_res_hex );
    if( ifca_ret_hex > rc )
      rc = ifca_ret_hex;
  }

  /******************************************************************
```

```
  * if IFI return buffer was too small, output a message            *
  ******************************************************************/
  if( sperind4 != -1  &&  xs_bytes_hex != 0 )
  {
    printf( " *** WARNING: %d bytes were lost", xs_bytes_hex );
    printf( " because the IFI return area in stored\n" );
    printf( " ***          procedure DSN8ED2 is too small" );
    printf( " to accomodate this request.\n" );
    printf( " ***       ** Increase the IFI return area" );
    printf( " (RETURN_LEN) in DSN8ED2 and then\n" );
    printf( " ***          recompile/relink/rebind before" );
    printf( " resubmitting the request.\n" );
    if( rc < RETWRN )
      rc = RETWRN;
  }

  /*******************************************************************
  * output any data from the error message buffer                   *
  ******************************************************************/
  if( sperind5 != -1 )
  {
    printf( " *** ERROR: The following diagnostics were returned by" );
    printf(     " stored procedure DSN8ED2.\n \n" );
    for( j = 0; j < sp_err_buf.sp_err_blen; j++ )
      printf( "%c",sp_err_buf.sp_err_txt[j] );
    printf( "\n" );
    if( rc < RETSEV )
      rc = RETSEV;
  }

} /* end of send_DB2_command_to_sp() */


/*********************************************************************
**********************************************************************
** Write out the DB2 command that has been built from input records **
**********************************************************************
*********************************************************************/
echo_DB2_command()                                            /*proc*/
{
  short int  c;                       /* local ptr to command buffer*/
  short int  k,kk,l;                  /* counters and loop control  */

  printf( " \n \n *** Input Statement:\n" );

  c = 0;
  /*******************************************************************
  * calculate no. of full print lines the cmd uses and print them    *
  ******************************************************************/
  kk = cmdbuf.cmdlen / (OUTLEN - 1);
  for( k=1; k<=kk; k++ )
  {
    printf( " " );
    for( l=0; l<(OUTLEN-1); l++ )
    {
      printf( "%c",cmdbuf.cmdtxt[c++] );
    }
    printf( "\n" );
  }

  /*******************************************************************
  * calculate no. of partial print lines the cmd uses; print them    *
  ******************************************************************/
  kk = cmdbuf.cmdlen % (OUTLEN - 1);
  if( kk > 0 )
  {
    printf( " " );
    for( k=1; k<=kk; k++ )
    {
      printf( "%c",cmdbuf.cmdtxt[c++] );
    }
    printf( "\n" );
  }
} /* end of echo_DB2_command() */


/*********************************************************************
**********************************************************************
** Output the contents of the result set returned by the stored     **
** procedure.                                                       **
**********************************************************************
*********************************************************************/
```

```
output_results_from_sp()                                      /*proc*/
{

  /******************************************************************
  * local initialization                                            *
  ******************************************************************/
  for(j=0; j<(OUTLEN-1); j++)         /* Blank the input array      */
    rs_data[j] = BLANK;               /* Initialize result string   */
  rs_sequence = 0;                    /* Initialize data sequence   */

  printf( " \n \n *** IFI return area:\n" );

  /******************************************************************
  * associate a locator variable with the result                    *
  ******************************************************************/
  EXEC SQL ASSOCIATE LOCATOR          /* Associate the result set   */
    (:DSN8ED2_rs_loc)                 /* locator with a host var.   */
    WITH PROCEDURE DSN8.DSN8ED2;      /*                            */
  if (SQLCODE != 0 )                  /* If unsuccessful then       */
  {                                   /* - Say so                   */
    sql_error( "*** Associate result set locator call unsuccessful." );
                                      /* - Print the sqlcode        */
    rc = RETSEV;                      /* - Flush remainder of proc  */
  }                                   /*                            */

  /******************************************************************
  * allocate the result set cursor                                  *
  ******************************************************************/
  if( rc < RETSEV )                   /* Or if okay so far then     */
  {                                   /*                            */
    EXEC SQL ALLOCATE DSN8ED2_RS_CSR  /* - Allocate a cursor to read*/
      CURSOR FOR                      /* - Allocate a cursor to read*/
      RESULT SET :DSN8ED2_rs_loc;     /*   the result set locator   */
    if (SQLCODE != 0 )                /* - If unsuccessful then     */
    {                                 /*   - Say so                 */
      sql_error( "*** Allocate result set cursor call unsuccessful." );
                                      /*   - Print the sqlcode      */
      rc = RETSEV;                    /*   - Flush remainder of proc*/
    }                                 /*                            */
  }                                   /*                            */

  /******************************************************************
  * fetch first row from the result set                             *
  ******************************************************************/
  if( rc < RETSEV )                   /* Or if okay so far then     */
  {
    EXEC SQL FETCH DSN8ED2_RS_CSR     /* - Fetch first row (if any) */
      INTO :rs_sequence, :rs_data;    /*   from the result set csr  */
    if (SQLCODE != 0 )                /* - If unsuccessful then     */
    {                                 /*   - Say so                 */
      sql_error("*** Priming fetch of result set cursor unsuccessful");
                                      /*   - Print the sqlcode      */
      rc = RETSEV;                    /*   - Flush remainder of proc*/
    }                                 /*                            */
  }                                   /*                            */

  /******************************************************************
  * output the contents of the result set                           *
  ******************************************************************/
  while(SQLCODE == 0 && rc < RETSEV)  /* Or if okay so far then     */
  {                                   /* until all lines processed  */
    printf( " %s\n", rs_data );       /* -- Output current line     */
                                      /*                            */
    EXEC SQL FETCH DSN8ED2_RS_CSR     /* -- Get the next one from   */
      INTO :rs_sequence, :rs_data;    /*    the result set cursor   */
  }                                   /*                            */

  /******************************************************************
  * check for successful processing of result set                   *
  ******************************************************************/
  if (SQLCODE != 100 && rc < RETSEV)  /* If unsuccessful then       */
  {                                   /* - Say so                   */
    sql_error( "*** Fetch of result set cursor unsuccessful." );
                                      /* - Print the sqlcode        */
    rc = RETSEV;                      /* - Set return code          */
  }                                   /*                            */

} /* end of output_results_from_sp() */


/********************************************************************
*********************************************************************
```

```
** SQL error handler                                              **
*********************************************************************
********************************************************************/
#pragma linkage(dsntiar, OS)

sql_error( char locmsg[] )                                   /*proc*/
{
  #define     DATA_DIM       10       /* Number of message lines    */

  struct      error_struct {          /* DSNTIAR message structure  */
    short int   error_len;
    char        error_text[DATA_DIM][OUTLEN-1];
    }         error_message = {DATA_DIM * (OUTLEN-1)};

  extern short int dsntiar( struct      sqlca         *sqlca,
                            struct      error_struct  *msg,
                            int                       *len );

  short int   rc;                     /* DSNTIAR Return code        */
  int         j;                      /* Loop control               */
  static int  lrecl = OUTLEN - 1;     /* Width of message lines     */

  /*****************************************************************
  * print the locator message                                       *
  *****************************************************************/
  printf( " %.80s\n", locmsg );

  /*****************************************************************
  * format and print the SQL message                                *
  *****************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<=DATA_DIM; j++ )
      printf( " %.80s\n", error_message.error_text[j] );
  else
    {
      printf( " *** ERROR: DSNTIAR could not format the message\n" );
      printf( " ***        SQLCODE is %d\n",SQLCODE );
      printf( " ***        SQLERRM is \n" );
      for( j=0; j<sqlca.sqlerrml; j++ )
        printf( "%c", sqlca.sqlerrmc[j] );
      printf( "\n" );
    }

} /* end of sql_error */
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8ED2

Use the Instrumentation Facility Interface (IFI) to process a Db2 command which has been passed from DSN8ED1, the requester program.

```
/*******************************************************************  00010000
*   Module name = DSN8ED2 (sample program)                          *  00020000
*                                                                   *  00030000
*   Descriptive name = Stored procedure result set server program   *  00040000
*                                                                   *  00050000
*     LICENSED MATERIALS - PROPERTY OF IBM                          *  00060000
*     5675-DB2                                                      *  00070000
*     (C) COPYRIGHT 1997, 2000 IBM CORP.  ALL RIGHTS RESERVED.      *  00080000
*                                                                   *  00090000
*     STATUS = VERSION 7                                            *  00100000
*                                                                   *  00110000
*   Function:                                                       *  00120000
*      Use the Instrumentation Facility Interface (IFI) to process  *  00130000
*      a DB2 command which has been passed from DSN8ED1, the        *  00140000
*      requester program.  Load the responses to a temporary DB2    *  00150000
*      table and return them as a result set.                       *  00160000
*                                                                   *  00170000
*   Notes:                                                          *  00180000
*      Dependencies =                                               *  00190000
*         1.  Must be linked and run under LE/370                   *  00200000
*         2.  Requires global temporary table DSN8.DSN8ED2_RS_TBL   *  00202990
*             (created by sample job DSNTEJ6T)                      *  00205980
*                                                                   *  00210000
```

```
*      Restrictions =                                                *
*         1.  The Instrumentation Facility Communication Area        *
*             (IFCA) contains information regarding the success      *
*             of the call and provides feedback.                     *
*             This area must be maintained to include any changes    *
*             to the mapping macro DSNDIFCA.                         *
*                                                                    *
*         2. A command may be no more than 4096 bytes.               *
*                                                                    *
*   Module type  = C program                                         *
*      Processor =                                                   *
*         ADMF precompiler                                           *
*         C/370                                                      *
*      Module size = See linkedit output                             *
*      Attributes = Not re-entrant nor re-usable                     *
*                                                                    *
*   Entry Point = CEESTART (LE/370)                                  *
*      Purpose  = See function                                       *
*      Linkage  = SIMPLE WITH NULLS                                  *
*                 Invoked via EXEC SQL call                          *
*      Input    = Parameters explicitly passed to this function:     *
*         Symbolic label/name = ARGV[1] (puts inputcmd)              *
*         Description = DB2 command to be processed by IFI.          *
*                       Input statements from this parameter         *
*                       will be passed to the text_or_command        *
*                       field of the output_area of the IFI          *
*                       utility for processing.                      *
*                                                                    *
*      Output   = Parameters explicitly returned:                    *
*         Symbolic label/name = ARGV[2] (gets ifca_ret_hex)          *
*         - IFI return code, in hex                                  *
*         Symbolic label/name = ARGV[3] (gets ifca_res_hex)          *
*         - IFI reason code, in hex                                  *
*         Symbolic label/name = ARGV[4] (gets xs_bytes_hex)          *
*         - Excess bytes not returned, in hex                        *
*         Symbolic label/name = ARGV[5] (gets errmsg_buf)            *
*         - Formatted SQL error messages                             *
*         Symbolic label/name = ARGV[6] (gets indvar)                *
*         - DB2 indicator variables                                  *
*                                                                    *
*      Output   = Result set returned:                               *
*         Result set cursor name = DSN8ED2_RS_CSR                    *
*         - Formatted responses from IFI for input command           *
*                                                                    *
*   Exit normal =                                                    *
*      No errors were found in the passed DB2 command and no         *
*      errors occurred during processing.                            *
*                                                                    *
*   Normal messages =                                                *
*                                                                    *
*   Exit-error =                                                     *
*      Errors were found in the passed DB2 command or occurred       *
*      during processing.                                            *
*                                                                    *
*      Return codes: n/a                                             *
*                                                                    *
*   Error messages = see under output                                *
*                                                                    *
*   External references =                                            *
*      Routines/services = none                                      *
*      Data areas        = none                                      *
*      Control blocks    = none                                      *
*                                                                    *
*   Pseudocode      =                                                *
*     DSN8ED2: Main                                                  *
*     - get the passed DB2 command.                                  *
*     - calculate the return area size for command requests.         *
*     - allocate the requested return area.                          *
*     - format the output area with the requested command.           *
*     - issue the command request to IFI.                            *
*     - create the temporary table to hold the result set.           *
*       - call sql_error if an unexpected SQLCODE is encountered     *
*     - extract the responses from the IFI return buffer and         *
*       insert them to the result set table.                         *
*       - call sql_error if an unexpected SQLCODE is encountered     *
*     - open the cursor to the result set table and exit.            *
*       - call sql_error if an unexpected SQLCODE is encountered     *
*     End DSN8ED2                                                    *
*                                                                    *
*     sql_error                                                      *
*     - invoke DSNTIAR to format the current SQL code and put the    *
*       messages to output parameter ARGV[5].                        *
```

```
*     - if DSNTIAR cannot detail the code, put the SQLCODE and the  *
*       SQLERRM to output parameter ARGV[5].                        *
*     End sql_error                                                 *
*******************************************************************/


/********************** C library definitions **********************/
#pragma     runopts( plist(mvs) )
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#pragma     linkage( dsnwli,OS )

/**************************** Constants ****************************/
#define     BLANK        ' '         /* Buffer padding              */
#define     BUFROWLN      80         /* Length of a report line     */
#define     DATA_DIM      10         /* Number of message lines     */
#define     HYPHEN       '-'         /* Hyphen                      */
#define     LINEFEED     '\n'        /* Linefeed character          */
#define     NULLCHAR     '\0'        /* Null character              */
#define     RETSEV        12         /* Severe error return code    */
#define     RETURN_LEN   8320        /* Length of IFI return buffer*/

/********************** Program Argument List **********************/
struct      inp  {                   /* Arg1 (in): Command stmt     */
  short int   incmlen;               /* - Input stmt length         */
  char        incmtxt[4096];         /* - Input stmt text           */
}           inputcmd;                /*                             */
struct inp  *inpptr;                 /* Pointer to input struct     */

long int    ifca_ret_hex;            /* Arg2 (out): IFI return code*/
long int    ifca_res_hex;            /* Arg3 (out): IFI reason code*/

long int    xs_bytes_hex;            /* Arg4 (out): # records lost */

char        errmsg[DATA_DIM+1][BUFROWLN];
                                     /* Arg5 (out): error messages */

short int   locind[5];               /* Arg6 (out): indicator vars */

/************************ Working variables ************************/
char        *parg1[];                /* Pointer to argument 1       */
int         *parg2;                  /* Pointer to argument 2       */
int         *parg3;                  /* Pointer to argument 3       */
int         *parg4;                  /* Pointer to argument 4       */
char        *parg5[][BUFROWLN];      /* Pointer to argument 5       */
short int   *parg6;                  /* Pointer to argument 6       */

long int    rc, lastrc;              /* Return codes                */

/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
long int    rs_sequence;             /* Result set data sequence    */
char        rs_data[81];             /* Result set data buffer      */
                                     /* - length is BUFROWLN+1 for */
                                     /*   C NUL-terminator byte)    */
EXEC SQL END DECLARE SECTION;

/******************** DB2 SQL Communication Area *******************/
EXEC SQL INCLUDE SQLCA;

/******************** DB2 SQL Cursor Declarations ******************/
EXEC SQL DECLARE  DSN8ED2_RS_CSR
          CURSOR  WITH RETURN WITH HOLD FOR
          SELECT  RS_SEQUENCE, RS_DATA
            FROM  DSN8.DSN8ED2_RS_TBL /* <- Created in job DSNTEJ6T */
        ORDER BY  RS_SEQUENCE;

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
int main( int argc, char *argv[] )    /* Argument count and list    */
{
  /*************************** Constants ***************************/

  char        eye[4]                 /* Const for IFI eye catcher  */
              = {'I','F','C','A'};
  char        loc[4]                 /* Const for IFI location     */
              = {'L','O','C','2'};

  /********************** Working variables ************************/
```

```
short int   numfull;                /* No. of lines in area passed*/
                                    /* from IFI that have BUFROWLN*/
                                    /* or more bytes              */
short int   partrow;                /* No. of lines in area passed*/
                                    /* from IFI that have less    */
                                    /* than BUFROWLN bytes        */

short int   i, j, k;                /* Loop control vars          */

char        *curbyte;               /* Pointer to current byte in */
                                    /*  return area               */

short int   len_bin;                /* Length of buffer, in binary*/
char        lenbyt1;                /* 1st byte of length         */
char        lenbyt2;                /* 2nd byte of length         */


/*********************** IFI Argument List ***********************/

char        function[9];            /* First parm for IFI call    */

/*****************************************************************
* IFCA - (Instrumentation Facility Communication Area) contains    *
*        information regarding the success of the call to IFI and  *
*        provides feedback information to the application program. *
*                                                                  *
* WARNING: This area must be maintained to include any changes to  *
*          the mapping macro DSNDIFCA.                             *
*****************************************************************/

typedef struct {                    /* Second parm for IFI call   */
  short int   lngth;                /* Length of the IFCA,        */
                                    /*  including length field    */
  short int   unused1;              /* Reserved                   */
  char        eye_catcher[4];       /* Valid eye catcher of IFCA  */
                                    /*  used to verify IFCA block */
  char        owner_id[4];          /* Used to establish ownership*/
                                    /*  of an OPN destination     */
  long int    ifcarc1;              /* Rtrn code for IFC API call */
  long int    ifcarc2;              /* Reason cd for IFC API call */
  long int    bytes_moved;          /* Bytes of recrd rtrnd by IFI*/
  long int    excess_bytes;         /* Bytes that did not fit     */
  long int    opn_writ_seq_num;     /* Last OPN writer sequ numbr */
                                    /*  rtrnd for a READA function*/
  long int    num_recds_lost;       /* Records lost indicator     */
  char        opn_name_for_reada[4];/* OPN nm used for READA requ */
  struct {                          /* Area with up to 8 OPN names*/
    short int   opn_lngth;          /* Length+4 of OPN names rtrnd*/
    short int   unused2;            /* Reserved                   */
    char        array_opn_names[4][8];
                                    /* Area for OPN names returned*/
  }           opn_names_area;
  struct {                          /* Area with up to 8 trace nos*/
    short int   trace_lngth;        /* Length+4 of trace nos rtrnd*/
    short int   unused3;            /* Reserved                   */
    char        array_trace_nos[2][8];
                                    /* Area for trace nos returned*/
  }           trace_nos_area;
  struct {                          /* Diagnosticd area           */
    short int diagnos_lngth;        /* Diagnostics length         */
    short int unused4;              /* Reserved                   */
    char diagnos_data[80];          /* Diagnostics data           */
  }           diagnos_area;
}           ifca;                   /****** end IFCA typedef ******/

ifca        *pi;                    /* Pointer to IFCA structure  */

typedef struct {                    /* Third parm for IFI call    */
  short int   lngth;                /* Length+4 of text or command*/
  short int   unused;               /* Reserved                   */
  char        text_or_command[254]; /* Actual cmd or record text  */
  }         output_area;

output_area *po;                    /* Pointer to IFI output area */

typedef struct {                    /* Fourth parm for IFI call   */
  long int    lngth;                /* Length+4 of IFI return area*/
  char        rtrn_buff[RETURN_LEN];/* IFI return area            */
  }         return_area;

return_area *pr;                    /* Pointer to IFI return area */
```

```
/******************************************************************
* initialize working variables                                    *
******************************************************************/
rc = 0;                             /* Initialize return code     */
lastrc = 0;                         /* Initialize return code     */

for( i=0; i<DATA_DIM+1; i++ )       /* clear error message buffer */
  for( j=0; j<BUFROWLN; j++ )
    errmsg[i][j] = BLANK;

/******************************************************************
* get input parameter (command for IFI) from caller               *
******************************************************************/
parg1[1] = argv[1];                 /* Command text from caller   */
curbyte = parg1[1];                 /* Get pointer to input struct*/

/******************************************************************
* determine the length of the command text                        *
******************************************************************/
inputcmd.incmlen = 0;
i = 0;
while( *(curbyte) != NULLCHAR   &&    i < 4096 )
  {
    inputcmd.incmtxt[i] = *curbyte;
    i++;
    curbyte++;
    inputcmd.incmlen++;
  }

/******************************************************************
* initialize the IFI parameters                                   *
******************************************************************/
strncpy( function,"COMMAND \0",9 ); /* Set constant               */

pi = malloc( sizeof(ifca) );        /* Point to IFCA structure    */
pi->lngth = sizeof(ifca);           /* Note length of IFCA area   */
for(i=0; i<4; i++ )
{
 pi->eye_catcher[i] = eye[i];       /* Initialize eye catcher     */
 pi->owner_id[i] = loc[i];          /* DB2 Loc: 1=Local, 2=Remote */
}

pr = malloc( sizeof(return_area) ); /* Point to IFI return area   */
for( i=0; i<RETURN_LEN; i++ )
  pr->rtrn_buff[i] = BLANK;         /* Clear the return buffer    */
pr->lngth = RETURN_LEN;             /* Length of return buffer    */

po = malloc( sizeof(output_area) ); /* Point to IFI command area  */
po->lngth = inputcmd.incmlen+4;     /* Note length of command text*/
for( i=0; i<254; i++ )              /* Copy in command            */
  po->text_or_command[i] = inputcmd.incmtxt[i];

/******************************************************************
* make the IFI call via the DSNWLI macro                          *
******************************************************************/
dsnwli( function,pi,pr,po );

/******************************************************************
* copy IFI command status codes to output parms                   *
******************************************************************/
ifca_ret_hex = pi->ifcarc1;         /* IFI Return code in binary  */
ifca_res_hex = pi->ifcarc2;         /* IFI Reason code in binary  */
xs_bytes_hex = pi->excess_bytes;    /* Bytes that did not fit     */

/******************************************************************
* Extract records from the IFI return area and place them in a    *
* table for transmission to the caller via a result set           *
******************************************************************/
if( pi->bytes_moved != 0 )          /* If data was returned by IFI*/
{
  /****************************************************************
  * First, clear any residue from the result set table            *
  ****************************************************************/
  EXEC SQL DELETE
             FROM  DSN8.DSN8ED2_RS_TBL;
  if( SQLCODE != 0                  /* 0 because everything is ok */
    & SQLCODE != +88 )              /* +88 because all rows del'd */
    sql_error( "*** SQL error when clearing temp table ..." );

  rs_sequence = 0;                  /* Init result set sequence no*/
  for( k=0; k<BUFROWLN; k++ )       /* Clear result set data var  */
    rs_data[k] = BLANK;
```

```

/*****************************************************************
* The IFI return buffer contains one or more variable length     *
* records.  Each record consists of a 4-byte length component    *
* followed by a text component.  The length component contains   *
* the length of the text component plus 4 to account for its     *
* own length.                                                    *
******************************************************************
* Extract the length of the 1st record in the buffer and sub-    *
* tract 4 bytes to obtain the length of just the text portion.   *
*****************************************************************/
curbyte = &( pr->rtrn_buff[0] );  /* Point to 1st byte in buffer*/
lenbyt1 = *(curbyte);             /* Set 1st byte of length     */
lenbyt2 = *(curbyte+1);           /* Set 2nd byte of length     */

len_bin  = ( (short int)lenbyt1 ) * 10 + ( (short int)lenbyt2 );
len_bin = len_bin - 4;            /* Discount size of length fld*/

/*****************************************************************
* For each IFI record returned, create one or more records of    *
* length BUFROWLN and insert them to the result set table        *
*****************************************************************/
while( ( rc < RETSEV ) && (pi->bytes_moved - len_bin) > 2 )
{
  curbyte = curbyte + 4;          /* Update position in buffer  */

  if( ((short int)( curbyte - &(pr->rtrn_buff[0]) ) + len_bin - 1)
    > pi->bytes_moved )
    break;                        /* At end of buffer           */

  numfull = len_bin / BUFROWLN;   /* No. rows of BUFROWLN bytes */
  partrow = len_bin % BUFROWLN;   /* No. of bytes leftover      */

  /***************************************************************
  * Move all complete lines                                      *
  ***************************************************************/
  if( numfull > 0 )
    for( j=0; j<numfull; j++ )
    {
      for( i=0; i<BUFROWLN; i++ ) /* Clear result set tbl buffer*/
        rs_data[i] = BLANK;
      for( i=0; i<BUFROWLN; i++ )
      {
        rs_data[i] = *curbyte;    /* Build result set table rec */
        curbyte++;                /* Bump ptr into IFI rtrn buff*/
      }

      rs_sequence++;              /* Bump result set tbl sequ no*/
      EXEC SQL INSERT             /* Insert to the table        */
                 INTO  DSN8.DSN8ED2_RS_TBL
                     ( RS_SEQUENCE,RS_DATA )
                VALUES(:rs_sequence,:rs_data );
      if( SQLCODE != 0 )
        sql_error( "*** SQL error when inserting full line ..." );
    }

  /***************************************************************
  * Move leftover bytes to one last result set table record      *
  ***************************************************************/
  if( rc < RETSEV && partrow > 0 )
  {
    for( i=0; i<BUFROWLN; i++ )  /* Clear result set tbl buffer*/
      rs_data[i] = BLANK;
    for( i=0; i<partrow; i++ )
    {
      rs_data[i] = *curbyte;      /* Build result set table rec */
      curbyte++;                  /* Bump ptr into IFI rtrn buff*/
    }
    rs_data[i-1] = BLANK;         /* Discard linefeed char      */

    rs_sequence++;                /* Bump result set tbl sequ no*/

    EXEC SQL INSERT               /* Insert to the table        */
               INTO  DSN8.DSN8ED2_RS_TBL
                   ( RS_SEQUENCE,RS_DATA )
              VALUES(:rs_sequence,:rs_data );
    if( SQLCODE != 0 )
      sql_error( "*** SQL error when inserting partial line ..." );
  }                               /* End-move partial line      */

  /***************************************************************
  * Advance to next record in the IFI buffer, extract its length,*
```

```
      * and subtract 4 bytes to get length of text portion           *
      **************************************************************/
      lenbyt1 = *(curbyte);            /* Set 1st byte of length    */
      lenbyt2 = *(curbyte+1);          /* Set 2nd byte of length    */

      len_bin  = ( (short int)lenbyt1) * 10 + ((short int)lenbyt2 );
      len_bin = len_bin - 4;           /* Discount for length field */

    } /* End of copying IFI return text to result set table */

    /****************************************************************
    * Open the cursor to the result set table on the way out         *
    ****************************************************************/
    if( rc < RETSEV )
    {
      EXEC SQL OPEN  DSN8ED2_RS_CSR;
      if( SQLCODE != 0 )
        sql_error( "*** SQL error when opening result set cursor ...");
    }

  } /* End of if data was returned by IFI */

  /******************************************************************
  * Set output arguments and DB2 locator variables                   *
  ******************************************************************/
  parg2 = (int *)argv[2];              /* locate and recast 2nd arg */
  *parg2 = ifca_ret_hex;               /*  assign it ifca return cd */
  locind[1] = 0;                       /* tell DB2 to transmit it   */

  parg3 = (int *)argv[3];              /* locate and recast 3rd arg */
  *parg3 = ifca_res_hex;               /*  assign it ifca reason cd */
  locind[2] = 0;                       /* tell DB2 to transmit it   */

  parg4 = (int *)argv[4];              /* locate and recast 4th arg */
  *parg4 = xs_bytes_hex;               /*  and assign it bytes lost */
  locind[3] = 0;                       /* tell DB2 to transmit it   */

  if( errmsg[0][0] == BLANK )          /* if no error message exists*/
    locind[4] = -1;                    /* -tell DB2 not to send one */
  else                                 /* otherwise copy it over and*/
  {                                    /* tell DB2 to transmit it   */
    parg5[0][0] = argv[5];             /* -locate the 5th func arg  */
    curbyte = parg5[0][0];             /* -set helper pointer       */
    for( i=0; i<DATA_DIM+1; i++ )      /* -parse a row, looking for */
    {                                  /*  the end of its msg text  */
      j = 0;
      while( errmsg[i][j] != NULLCHAR  &&  j < BUFROWLN )
      {
        *curbyte = errmsg[i][j++];     /* -copy nonnull bytes       */
        curbyte++;
      }
      errmsg[i][j] = LINEFEED;         /* -add linefd to end of row */
    } /* End of for( i=0; i<DATA_DIM+1; i++ ) */

    *curbyte = NULLCHAR;               /* -null-terminate the buffer*/
    locind[4] = 0;                     /* -tell DB2 to transmit it  */
  } /* End of if( errmsg[0][0] != BLANK ) */

  parg6 = (short int *)argv[6];        /* locate and recast 6th arg */
  for( j=0; j<5; j++ )                 /* copy over null-ind array  */
  {
    *parg6 = locind[j];
    parg6++;
  }                                    /* return control to caller  */

} /* end of main */

/********************************************************************
* SQL error handler                                                  *
********************************************************************/
#pragma linkage(dsntiar, OS)

sql_error( char locmsg[] )
{
                                     /* DSNTIAR message structure  */
  struct      error_struct {
    short int   error_len;
    char        error_text[DATA_DIM][BUFROWLN];
    }         error_message = {DATA_DIM * BUFROWLN};

  extern short int dsntiar( struct      sqlca         *sqlca,
```

```
                              struct      error_struct  *msg,
                              int                       *len );

    char        *curbyte;               /* Pointer to current byte in */
                                        /*  error_message             */

    short int   tiar_rc;                /* DSNTIAR Return code        */
    int         i;                      /* Loop control               */
    static int  lrecl = BUFROWLN;       /* Width of message lines     */

    /*****************************************************************
    * indicate that a fatal error has occurred                       *
    *****************************************************************/
    rc = RETSEV;

    /*****************************************************************
    * copy locator message to the error message return buffer        *
    *****************************************************************/
    strcpy( errmsg[0],locmsg );

    /*****************************************************************
    * format the SQL message and move it to the err msg rtn buffer   *
    *****************************************************************/
    tiar_rc = dsntiar( &sqlca, &error_message, &lrecl );

    if( tiar_rc == 0 )
      for( i=0; i<DATA_DIM; i++ )
      {
        strncpy( errmsg[i+1],error_message.error_text[i],BUFROWLN );
      }
    else
      {
        strcpy( errmsg[1],"DSNTIAR could not detail the SQL error" );
        strcpy( errmsg[2],"*** SQLCODE is " );
        strcat( errmsg[3],(char *)SQLCODE );
        strcpy( errmsg[4],"*** SQLERRM is " );
        for( i=0; i<sqlca.sqlerrml; i++ )
          errmsg[5][i],sqlca.sqlerrmc[i];
      }

 } /* end of sql_error */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EC1

Demonstrates how a Db2 stored procedure can use IMS Open Database Access (ODBA) to connect to IMS DBCTL and access IMS data.

```
 CBL  APOST,LIST,RENT
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  DSN8EC1

      ****** DSN8EC1 - DB2 Sample ODBA Stored Procedure ***************
      *                                                               *
      *   Module Name = DSN8EC1                                       *
      *                                                               *
      *   Descriptive Name = DB2 Sample Application                   *
      *                      DB2 Sample ODBA Stored Procedure         *
      *                      Batch                                    *
      *                      Cobol                                    *
      *                                                               *
      *LICENSED MATERIALS - PROPERTY OF IBM                           *
      *5675-DB2                                                       *
      *(C) COPYRIGHT 1999, 2000 IBM CORP.  ALL RIGHTS RESERVED.       *
      *                                                               *
      *STATUS = VERSION 7                                             *
      *                                                               *
      *   Function = Demonstrates how a DB2 stored procedure can use  *
      *              IMS Open Database Access (ODBA) to connect to    *
      *              IMS DBCTL and access IMS data.                   *
      *                                                               *
      *              In particular, this program allows its client    *
      *              to add, retrieve, update, and delete entries in  *
      *              the IMS IVP telephone directory database,        *
      *              DSNIVD1.                                         *
```

```
      *                                                               * 00002800
      *                                                               * 00002900
      *   Notes = The following conditions must be satisfied:        * 00003000
      *     (1) DSN8EC1 is registered in DB2 on a server that also    * 00003100
      *         has an IMS subsystem operating at IMS/ESA V6 or a     * 00003200
      *         subsequent release (required for ODBA).               * 00003300
      *     (2) The following IMS IVP parts are available on that IMS * 00003400
      *         subsystem:                                            * 00003500
      *         (1) DFSIVD1, the IMS IVP telephone directory database * 00003600
      *         (2) DFSIVP64, the IMS IVP Cobol PSB for BMP access to * 00003700
      *             DFSIVD1                                           * 00003800
      *     (3) DSN8EC1 must be run a WLM-established stored proce-   * 00003900
      *         dures address space only                              * 00004000
      *     (4) The WLM environment associated with DSN8EC1 in SYSIBM.* 00004100
      *         SYSPROCEDURES is started by a proc that references    * 00004200
      *         the IMS reslib in both the STEPLIB DD concatenation   * 00004300
      *         and in the DFSRESLB DD.  See the DB2 Installation     * 00004400
      *         Guide for more information.                           * 00004500
      *                                                               * 00004600
      *   Module Type = Cobol Program                                 * 00004700
      *      Processor   = DB2 for OS/390 precompiler, IBM Cobol      * 00004800
      *      Module Size = See linkedit output                        * 00004900
      *      Attributes  = Re-entrant                                 * 00005000
      *                                                               * 00005100
      *                                                               * 00005200
      *   Entry Point = DSN8EC1                                       * 00005300
      *      Purpose = See function                                   * 00005400
      *      Linkage = Standard MVS program invocation                * 00005500
      *      Input   = Parameters explicitly passed to this function: * 00005600
      *                TDBCTLID ...... PIC X(8)                       * 00005700
      *                - IMS subsystem id                             * 00005800
      *                COMMAND ....... PIC X(8)                       * 00005900
      *                - Action to perform: ADD, UPD, DIS, DEL        * 00006000
      *                LAST-NAME ..... PIC X(10)                      * 00006100
      *                FIRST-NAME .... PIC X(10)                      * 00006200
      *                EXTENSION ..... PIC X(10)                      * 00006300
      *                ZIP-CODE ...... PIC X(7)                       * 00006400
      *                                                               * 00006500
      *      Output  = Parameters explicitly passed by this function  * 00006600
      *                COMMAND ....... PIC X(8)                       * 00006700
      *                - Action performed: ADD, UPD, DIS, DEL         * 00006800
      *                LAST-NAME ..... PIC X(10)                      * 00006900
      *                FIRST-NAME .... PIC X(10)                      * 00007000
      *                EXTENSION ..... PIC X(10)                      * 00007100
      *                ZIP-CODE ...... PIC X(7)                       * 00007200
      *                AIBRETRN ...... PIC S9(9) COMP                 * 00007300
      *                - Return code from IMS AIB call                * 00007400
      *                AIBREASN ...... PIC S9(9) COMP                 * 00007500
      *                - Reason code from IMS AIB call                * 00007600
      *                ERROR-CALL .... PIC X(4)                       * 00007700
      *                - DL/I command that failed                     * 00007800
      *                                                               * 00007900
      *   Exit-Normal = Return Code 0 Normal Completion               * 00008000
      *                                                               * 00008100
      *   Exit-Error =  Return Code 0 Abnormal Completion             * 00008200
      *                                                               * 00008300
      *      Error Messages = None: Errors are signaled by means of   * 00008400
      *                       SQLCODEs and DL/I codes returned to the * 00008500
      *                       client.                                 * 00008600
      *                                                               * 00008700
      *   External References =                                       * 00008800
      *      Routines/Services =                                      * 00008900
      *            AERTDLI  -     DL/I interface for ODBA             * 00009000
      *                                                               * 00009100
      *      Data areas        =  None                                * 00009200
      *                                                               * 00009300
      *      Control Blocks    =                                      * 00009400
      *            AIB      -     DL/I Application Interface Block    * 00009500
      *                                                               * 00009600
      *   Tables =  None                                              * 00009700
      *                                                               * 00009800
      *                                                               * 00009900
      *   Change Activity = None                                      * 00010000
      *                                                               * 00010100
      *                                                               * 00010200
      *  *Pseudocode*                                                 * 00010300
      *                                                               * 00010400
      *  PROCEDURE A00000-ODBA-SP                                     * 00010500
      *    Call B10000-ALLOCATE-AIB to allocate the IMS AIB           * 00010600
      *    Call B20000-PREPARE-REQUEST to format input from the client* 00010700
      *    Call B30000-PROCESS-REQUEST to access data on IMS          * 00010800
      *         Call C31000-ADD-ENTRY if client passed ADD request    * 00010900
```

```
*              Call D31100-INSERT-TO-DB to process IMS ISRT     *
*         Call C32000-UPDATE-ENTRY if client passed UPD request *
*              Call D32100-GET-HOLD-UNIQUE-FROM-DB for IMS GHU  *
*              Call D32200-REPLACE-IN-DB for IMS REPL           *
*         Call C33000-DELETE-ENTRY if client passed DEL request *
*              Call D32100-GET-HOLD-UNIQUE-FROM-DB for IMS GHU  *
*              Call D33200-DELETE-FROM-DB for IMS DLET          *
*         Call C34000-DISPLAY-ENTRY if client passed DIS request*
*              Call D34100-GET-UNIQUE-FROM-DB for IMS GU        *
*    Call B40000-DEALLOCATE-AIB to pend unit of work on IMS     *
*                                                               *
*---------------------------------------------------------------*



 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-370.
 OBJECT-COMPUTER.  IBM-370.

 INPUT-OUTPUT SECTION.

 DATA DIVISION.
 WORKING-STORAGE SECTION.

*****************************************************************
* DL/I-related declarations
*****************************************************************
* Application Interface Block(AIB) mapping
 01  AIB.
     02 AIBID              PIC X(8).
     02 AIBLEN             PIC 9(9) USAGE BINARY.
     02 AIBSFUNC           PIC X(8).
     02 AIBRSNM1           PIC X(8).
     02 AIBRSNM2           PIC X(8).
     02 AIBRESV1           PIC X(8).
     02 AIBOALEN           PIC 9(9) USAGE BINARY.
     02 AIBOAUSE           PIC 9(9) USAGE BINARY.
     02 AIBRESV2           PIC X(12).
     02 AIBRETRN           PIC 9(9) USAGE BINARY.
     02 AIBREASN           PIC 9(9) USAGE BINARY.
     02 AIBRESV3           PIC X(4).
     02 AIBRESA1           USAGE POINTER.
     02 AIBRESA2           USAGE POINTER.
     02 AIBRESA3           USAGE POINTER.
     02 AIBRESV4           PIC X(40).
     02 AIBSAVE      OCCURS 18 TIMES
                     USAGE POINTER.
     02 AIBTOKN      OCCURS 6 TIMES
                     USAGE POINTER.
     02 AIBTOKC            PIC X(16).
     02 AIBTOKV            PIC X(16).
     02 AIBTOKA      OCCURS 2 TIMES
                     PIC 9(9) USAGE BINARY.

* Segment Search Argument (SSA)
 01 SSA.
    02  SEGMENT-NAME  PIC X(8)  VALUE 'A1111111'.
    02  SEG-KEY-NAME  PIC X(11) VALUE '(A1111111 ='.
    02  SSA-KEY       PIC X(10).
    02  FILLER        PIC X VALUE ')'.

* Initializers
 77  SSA1            PIC X(9)  VALUE 'A1111111 '.
 77  APSBNME         PIC X(8)  VALUE 'DFSIVP6'.
 77  DPCBNME         PIC X(8)  VALUE 'TELEPCB1'.
 77  VAIBID          PIC X(8)  VALUE 'DFSAIB  '.
 77  SFPREP          PIC X(4)  VALUE 'PREP'.

* DL/I function codes
 77  GET-UNIQUE      PIC  X(4)  VALUE 'GU  '.
 77  GET-HOLD-UNIQUE PIC  X(4)  VALUE 'GHU '.
 77  GET-NEXT        PIC  X(4)  VALUE 'GN  '.
 77  ISRT            PIC  X(4)  VALUE 'ISRT'.
 77  DLET            PIC  X(4)  VALUE 'DLET'.
 77  REPL            PIC  X(4)  VALUE 'REPL'.
 77  APSB            PIC  X(4)  VALUE 'APSB'.
 77  DPSB            PIC  X(4)  VALUE 'DPSB'.
 77  APPERR          PIC  X(3)  VALUE '264'.
 77  INVCMD          PIC  X(3)  VALUE '440'.
 77  NOKEY           PIC  X(3)  VALUE '218'.

```

```
      ******************************************************************
      * I/O area for datacase handling
      ******************************************************************
       01  IOAREA.
           02  IO-BLANK  PIC  X(37) VALUE SPACES.
           02  IO-DATA REDEFINES IO-BLANK.
               03  IO-LAST-NAME   PIC  X(10).
               03  IO-FIRST-NAME  PIC  X(10).
               03  IO-EXTENSION   PIC  X(10).
               03  IO-ZIP-CODE    PIC  X(7).
           02  IO-FILLER    PIC  X(3) VALUE SPACES.
           02  IO-COMMAND   PIC  X(8) VALUE SPACES.

       01  DB2IN-COMMAND.
           02  DB2IW-COMMAND    PIC  X(8).
           02  DB2TEMP-COMMAND REDEFINES DB2IW-COMMAND.
               03  DB2TEMP-IOCMD PIC  X(3).
               03  FILLER     PIC  X(5).

      ******************************************************************
      * Miscellaneous variables
      ******************************************************************
       77  TEMP-ONE   PICTURE X(8) VALUE SPACES.
       77  TEMP-TWO   PICTURE X(8) VALUE SPACES.
       77  REPLY      PICTURE X(16).

       01 FLAGS.
          02  SET-DATA-FLAG  PIC X VALUE '0'.
             88  NO-SET-DATA       VALUE '1'.
          02  TADD-FLAG      PIC X VALUE '0'.
             88  PROCESS-TADD      VALUE '1'.

       01 COUNTERS.
          02  L-SPACE-CTR    PIC   9(2) COMP VALUE 0.

       01 RUN-STATUS         PIC   X(4).
          88  NOT-OKAY                  VALUE 'BAD'.
          88  OKAY                      VALUE 'GOOD'.



       LINKAGE SECTION.

      ******************************************************************
      * Data area for DB2 Stored Procedures input/output
      ******************************************************************
       01  DB2IO-TDBCTLID   PIC  X(8).
       01  DB2IO-COMMAND    PIC  X(8).
       01  DB2IO-LAST-NAME  PIC  X(10).
       01  DB2IO-FIRST-NAME PIC  X(10).
       01  DB2IO-EXTENSION  PIC  X(10).
       01  DB2IO-ZIP-CODE   PIC  X(7).

      ******************************************************************
      * Data area for DB2 Stored Procedures output
      ******************************************************************
       01  DB2OUT-AIBRETRN      PIC S9(9) COMP.
       01  DB2OUT-AIBREASN      PIC S9(9) COMP.
       01  DC-ERROR-CALL        PIC X(4).


      ******************************************************************
      * Stored Procedure parameter list
      ******************************************************************
       PROCEDURE DIVISION
           USING DB2IO-TDBCTLID,
                 DB2IO-COMMAND,
                 DB2IO-LAST-NAME,
                 DB2IO-FIRST-NAME,
                 DB2IO-EXTENSION,
                 DB2IO-ZIP-CODE,
                 DB2OUT-AIBRETRN,
                 DB2OUT-AIBREASN,
                 DC-ERROR-CALL.


      ******************************************************************
      * Main Driver: Process data passed by client and apply the data
      *              to the IMS IVP phone book database, DFSIVD1.
      ******************************************************************
       A00000-ODBA-SP.
           MOVE 'GOOD' TO RUN-STATUS.
```

```
     PERFORM B10000-ALLOCATE-AIB.
     IF OKAY THEN
        PERFORM B20000-PREPARE-REQUEST.
     IF OKAY THEN
        PERFORM B30000-PROCESS-REQUEST.
     IF OKAY THEN
        PERFORM B40000-DEALLOCATE-AIB.

     STOP RUN.


*****************************************************************
* Initialize and allocate the Application Interface Block
*****************************************************************
 B10000-ALLOCATE-AIB.
     INITIALIZE AIB.
     SET  AIBRESA1 TO NULLS.
     SET  AIBRESA2 TO NULLS.
     SET  AIBRESA3 TO NULLS.
     MOVE ZEROES to AIBRETRN.
     MOVE ZEROES to AIBREASN.
     MOVE VAIBID to AIBID.
     MOVE LENGTH OF AIB to AIBLEN.
     MOVE SPACES to IOAREA.
     MOVE LENGTH OF IOAREA to AIBOALEN.
     MOVE SPACES TO AIBSFUNC.
     MOVE APSBNME to AIBRSNM1.
     MOVE DB2IO-TDBCTLID to AIBRSNM2.

*    Allocate the PSB for the AIB
     CALL 'AERTDLI' USING APSB, AIB.

     IF AIBRETRN EQUAL ZEROES THEN
        MOVE 0 TO SET-DATA-FLAG
        MOVE 0 TO TADD-FLAG
     ELSE
        MOVE 'BAD' TO RUN-STATUS
        MOVE AIBRETRN TO DB2OUT-AIBRETRN
        MOVE AIBREASN TO DB2OUT-AIBREASN.


*****************************************************************
* Prepare data passed from client for processing by ODBA
*****************************************************************
 B20000-PREPARE-REQUEST.

*    Check the leading space in input command and trim it off
     INSPECT DB2IO-COMMAND
       TALLYING L-SPACE-CTR FOR LEADING SPACE
       REPLACING LEADING SPACE BY '*'.
     IF L-SPACE-CTR > 0 THEN
        UNSTRING DB2IO-COMMAND
           DELIMITED BY ALL '*'
           INTO TEMP-ONE TEMP-TWO
        MOVE TEMP-TWO TO DB2IO-COMMAND
        MOVE 0 TO L-SPACE-CTR
        MOVE SPACES TO TEMP-TWO.

*    Check the leading space in input LAST NAME and trim it off
     INSPECT DB2IO-LAST-NAME
       TALLYING L-SPACE-CTR FOR LEADING SPACE
       REPLACING LEADING SPACE BY '*'.
     IF L-SPACE-CTR > 0 THEN
        UNSTRING DB2IO-LAST-NAME
          DELIMITED BY ALL '*'
          INTO TEMP-ONE TEMP-TWO
        MOVE TEMP-TWO TO DB2IO-LAST-NAME
        MOVE 0 TO L-SPACE-CTR
        MOVE SPACES TO TEMP-TWO.

*    Check the leading space in input FIRST NAME and trim it off
     INSPECT DB2IO-FIRST-NAME
       TALLYING L-SPACE-CTR FOR LEADING SPACE
       REPLACING LEADING SPACE BY '*'.
     IF L-SPACE-CTR > 0 THEN
        UNSTRING DB2IO-FIRST-NAME
          DELIMITED BY ALL '*'
          INTO TEMP-ONE TEMP-TWO
        MOVE TEMP-TWO TO DB2IO-FIRST-NAME
        MOVE 0 TO L-SPACE-CTR
        MOVE SPACES TO TEMP-TWO.

```

```
      *    Check the leading space in input EXTENSION and trim it off
           INSPECT DB2IO-EXTENSION
             TALLYING L-SPACE-CTR FOR LEADING SPACE
             REPLACING LEADING SPACE BY '*'.
           IF L-SPACE-CTR > 0 THEN
              UNSTRING DB2IO-EXTENSION
                DELIMITED BY ALL '*'
                INTO TEMP-ONE TEMP-TWO
              MOVE TEMP-TWO TO DB2IO-EXTENSION
              MOVE 0 TO L-SPACE-CTR
              MOVE SPACES TO TEMP-TWO.

      *    Check the leading space in input ZIP CODE and trim it off
           INSPECT DB2IO-ZIP-CODE
             TALLYING L-SPACE-CTR FOR LEADING SPACE
             REPLACING LEADING SPACE BY '*'.
           IF L-SPACE-CTR > 0 THEN
              UNSTRING DB2IO-ZIP-CODE
                DELIMITED BY ALL '*'
                INTO TEMP-ONE TEMP-TWO
              MOVE TEMP-TWO TO DB2IO-ZIP-CODE
              MOVE 0 TO L-SPACE-CTR
              MOVE SPACES TO TEMP-TWO.

      *    Move the data to IO area for IMS
           MOVE DB2IO-LAST-NAME TO IO-LAST-NAME.
           MOVE DB2IO-COMMAND TO IO-COMMAND.
           MOVE DB2IO-COMMAND TO DB2IN-COMMAND.
           DISPLAY 'TE>DB2TEMP-IOCMD=' DB2TEMP-IOCMD.

      *    If no command specified, issue error
           IF IO-COMMAND EQUAL SPACES THEN
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN

      *    If no LAST NAME specified, issue error
           ELSE IF IO-LAST-NAME EQUAL SPACES THEN
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR  TO DB2OUT-AIBRETRN
              MOVE NOKEY   TO DB2OUT-AIBREASN.


      *****************************************************************
      * Process the request from the client
      *****************************************************************
       B30000-PROCESS-REQUEST.

      *    If command is ADD, insert a new record
           IF DB2TEMP-IOCMD EQUAL 'ADD' THEN
              PERFORM C31000-ADD-ENTRY

      *    If command is TAD, insert a new record and trace with WTO
           ELSE IF DB2TEMP-IOCMD EQUAL 'TAD' THEN
              MOVE 1 TO TADD-FLAG
              PERFORM C31000-ADD-ENTRY

      *    If command is UPD, update existing record for LAST NAME
           ELSE IF DB2TEMP-IOCMD EQUAL 'UPD' THEN
              PERFORM C32000-UPDATE-ENTRY

      *    If command is DEL, delete record for LAST NAME
           ELSE IF DB2TEMP-IOCMD EQUAL 'DEL' THEN
              PERFORM C33000-DELETE-ENTRY

      *    If command is DIS, display record for LAST NAME
           ELSE IF DB2TEMP-IOCMD EQUAL 'DIS' THEN
              PERFORM C34000-DISPLAY-ENTRY

      *    Otherwise, issue error for unexpected command
           ELSE
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN.


      *****************************************************************
      * Deallocate the ODBA Application Interface Block
      *****************************************************************
       B40000-DEALLOCATE-AIB.
           MOVE APSBNME to AIBRSNM1.
      *   PREP keyword, below, tells IMS to move in-flight transactions
```

```
*   to in-doubt state, so checkpoint or rollback can be deferred
*   until DB2 stored procedure client issues COMMIT or ROLLBACK
     MOVE SFPREP to AIBSFUNC.

*   Deallocate the PSB for the AIB
     CALL 'AERTDLI' USING DPSB, AIB.
     DISPLAY 'AFTER DPSB PREP, DPCBNME=' DPCBNME.
     DISPLAY 'DPSB PREP AIBRETRN=' AIBRETRN.
     DISPLAY 'DPSB PREP AIBREASN=' AIBREASN.
     DISPLAY 'DPSB PREP AIBRSNM1=' AIBRSNM1.
     DISPLAY 'DPSB PREP AIBRSNM2=' AIBRSNM2.
     DISPLAY 'DPSB PREP AIBRESA1=' AIBRESA1.
     DISPLAY 'DPSB PREP AIBRESA2=' AIBRESA2.
     DISPLAY 'DPSB PREP AIBRESA3=' AIBRESA3.
     MOVE AIBRETRN TO DB2OUT-AIBRETRN.
     MOVE AIBREASN TO DB2OUT-AIBREASN.


*****************************************************************
* Addition request handler
*****************************************************************
 C31000-ADD-ENTRY.
     MOVE DB2IO-FIRST-NAME TO IO-FIRST-NAME.
     MOVE DB2IO-EXTENSION  TO IO-EXTENSION.
     MOVE DB2IO-ZIP-CODE   TO IO-ZIP-CODE.
     MOVE IO-COMMAND       TO DB2IO-COMMAND.

     IF DB2IO-FIRST-NAME EQUAL SPACES
       OR DB2IO-EXTENSION EQUAL SPACES
       OR DB2IO-ZIP-CODE EQUAL SPACES THEN
        MOVE 'BAD' TO RUN-STATUS
        MOVE APPERR TO DB2OUT-AIBRETRN
        MOVE INVCMD TO DB2OUT-AIBREASN
     ELSE
        PERFORM D31100-INSERT-TO-DB.


*****************************************************************
* Update request handler
*****************************************************************
 C32000-UPDATE-ENTRY.
     MOVE 0 TO SET-DATA-FLAG.
     MOVE IO-LAST-NAME TO SSA-KEY.
     PERFORM D32100-GET-HOLD-UNIQUE-FROM-DB.
     IF AIBRETRN = ZEROES THEN
        IF DB2IO-FIRST-NAME NOT = SPACES THEN
           MOVE 1 TO SET-DATA-FLAG
           MOVE DB2IO-FIRST-NAME TO IO-FIRST-NAME
        END-IF
        IF DB2IO-EXTENSION  NOT = SPACES THEN
           MOVE 1 TO SET-DATA-FLAG
           MOVE DB2IO-EXTENSION  TO IO-EXTENSION
        END-IF
        IF DB2IO-ZIP-CODE   NOT = SPACES THEN
           MOVE 1 TO SET-DATA-FLAG
           MOVE DB2IO-ZIP-CODE   TO IO-ZIP-CODE
        END-IF
        MOVE IO-COMMAND TO DB2IO-COMMAND.
     IF NO-SET-DATA THEN
        PERFORM D32200-REPLACE-IN-DB
     ELSE
        MOVE 'BAD' TO RUN-STATUS
        MOVE APPERR TO DB2OUT-AIBRETRN
        MOVE INVCMD TO DB2OUT-AIBREASN.


*****************************************************************
* Delete request handler
*****************************************************************
 C33000-DELETE-ENTRY.
     MOVE IO-LAST-NAME TO SSA-KEY.
     PERFORM D32100-GET-HOLD-UNIQUE-FROM-DB.
     IF AIBRETRN = ZEROES THEN
        MOVE IO-COMMAND TO DB2IO-COMMAND
        PERFORM D33200-DELETE-FROM-DB.


*****************************************************************
* Display request handler
*****************************************************************
 C34000-DISPLAY-ENTRY.
     MOVE IO-LAST-NAME TO SSA-KEY.
```

```
           DISPLAY 'TE>SSA-KEY=' SSA-KEY.
           PERFORM D34100-GET-UNIQUE-FROM-DB.
           IF AIBRETRN = ZEROES THEN
              MOVE IO-LAST-NAME TO DB2IO-LAST-NAME
              MOVE IO-FIRST-NAME TO DB2IO-FIRST-NAME
              MOVE IO-EXTENSION TO DB2IO-EXTENSION
              MOVE IO-ZIP-CODE TO DB2IO-ZIP-CODE
              MOVE IO-COMMAND TO DB2IO-COMMAND.
      *****************************************************************
      * Data base segment insert request handler
      *****************************************************************
       D31100-INSERT-TO-DB.
           MOVE DPCBNME to AIBRSNM1.
           CALL 'AERTDLI' USING ISRT, AIB, IOAREA, SSA1.
           IF AIBRETRN = ZEROES THEN
              IF PROCESS-TADD THEN
                 DISPLAY 'INSERT IS DONE, REPLY' UPON CONSOLE
                 ACCEPT REPLY FROM CONSOLE
                 MOVE 0 TO TADD-FLAG
              END-IF
           ELSE
              MOVE 'BAD' TO RUN-STATUS
              DISPLAY 'ISRT AIBRETRN=' AIBRETRN
              DISPLAY 'ISRT AIBREASN=' AIBREASN
              DISPLAY 'ISRT AIBRESA1=' AIBRESA1
              DISPLAY 'ISRT AIBRESA2=' AIBRESA2
              DISPLAY 'ISRT AIBRESA3=' AIBRESA3
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN
              MOVE ISRT TO DC-ERROR-CALL.
      *****************************************************************
      * Data base segment request handler
      *****************************************************************
       D32100-GET-HOLD-UNIQUE-FROM-DB.
           MOVE DPCBNME to AIBRSNM1.
           CALL 'AERTDLI' USING GET-HOLD-UNIQUE, AIB, IOAREA, SSA.
           IF AIBRETRN NOT EQUAL ZEROES THEN
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN
              MOVE GET-HOLD-UNIQUE TO DC-ERROR-CALL.
      *****************************************************************
      * Data base segment replace request handler
      *****************************************************************
       D32200-REPLACE-IN-DB.
           MOVE DPCBNME to AIBRSNM1.
           CALL 'AERTDLI' USING REPL, AIB, IOAREA.
           IF AIBRETRN NOT EQUAL ZEROES THEN
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN
              MOVE REPL TO DC-ERROR-CALL.
      *****************************************************************
      * Data base segment delete request handler
      *****************************************************************
       D33200-DELETE-FROM-DB.
           MOVE DPCBNME to AIBRSNM1.
           CALL 'AERTDLI' USING DLET, AIB, IOAREA.
           IF AIBRETRN NOT EQUAL ZEROES THEN
              MOVE 'BAD' TO RUN-STATUS
              MOVE APPERR TO DB2OUT-AIBRETRN
              MOVE INVCMD TO DB2OUT-AIBREASN
              MOVE DLET TO DC-ERROR-CALL.
      *****************************************************************
      * Data base segment GET-UNIQUE request handler
      *****************************************************************
       D34100-GET-UNIQUE-FROM-DB.
           MOVE DPCBNME to AIBRSNM1.
           CALL 'AERTDLI' USING GET-UNIQUE, AIB, IOAREA, SSA.
           IF AIBRETRN NOT EQUAL ZEROES THEN
              MOVE 'BAD' TO RUN-STATUS
              DISPLAY 'GU AIBRETRN=' AIBRETRN
```

```
           DISPLAY 'GU AIBREASN=' AIBREASN
           DISPLAY 'GU AIBRESA1(ADDR PCB)=' AIBRESA1
           DISPLAY 'GU AIBRESA2=' AIBRESA2
           DISPLAY 'GU AIBRESA3=' AIBRESA3
           MOVE APPERR TO DB2OUT-AIBRETRN
           MOVE INVCMD TO DB2OUT-AIBREASN
           MOVE GET-UNIQUE TO DC-ERROR-CALL.
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EC2

Demonstrates how to CALL the Db2 sample ODBA stored procedure, DSN8.

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. DSN8EC2.

       ****** DSN8EC2 - DB2 Sample ODBA Stored Procedure Client ********
       *                                                               *
       *   Module Name = DSN8EC2                                       *
       *                                                               *
       *   Descriptive Name = DB2 Sample Application                   *
       *                      Client for DB2 Sample ODBA Stored Proc   *
       *                      Batch                                    *
       *                      Cobol                                    *
       *                                                               *
       *LICENSED MATERIALS - PROPERTY OF IBM                           *
       *5675-DB2                                                       *
       *(C) COPYRIGHT 1999, 2000 IBM CORP.  ALL RIGHTS RESERVED.       *
       *                                                               *
       *STATUS = VERSION 7                                             *
       *                                                               *
       *   Function = Demonstrates how to CALL the DB2 sample ODBA     *
       *              stored procedure, DSN8.DSN8EC1, for accessing    *
       *              the IMS IVP telephone directory database,        *
       *              DFSIVD1.                                         *
       *                                                               *
       *              In particular, this program:                     *
       *              (1) Calls DSN8.DSN8EC1, passing an add request   *
       *                  and the data for an entry to be inserted to  *
       *                  DFSIVD1.                                     *
       *              (2) Commits the unit of work for both DB2 and    *
       *                  IMS (Note: IMS work is in an "in doubt"      *
       *                  status until the stored procedure client     *
       *                  performs a COMMIT or a ROLLBACK).            *
       *              (3) Calls DSN8.DSN8EC1 again, passing a display  *
       *                  request for a entry to be retrieved from     *
       *                  DFSIVD1.                                     *
       *                                                               *
       *                                                               *
       *   Notes = NONE                                                *
       *                                                               *
       *   Module Type = Cobol Program                                 *
       *      Processor   = DB2 for OS/390 precompiler, IBM Cobol      *
       *      Module Size = See linkedit output                        *
       *      Attributes  = Re-entrant                                 *
       *                                                               *
       *                                                               *
       *   Entry Point = DSN8EC2                                       *
       *      Purpose = See function                                   *
       *      Linkage = Standard MVS program invocation                *
       *                                                               *
       *      Input   = Parameters explicitly passed to this function: *
       *                PARMS ......... PIC X(25)                      *
       *                                                               *
       *      Output  = Symbolic label/Name = SYSOUT                   *
       *                Description         = Results of ADD and DIS   *
       *                                                               *
       *   Exit-Normal = Return Code 0 Normal Completion               *
       *                                                               *
       *   Exit-Error =  Return Code 8 Abnormal Completion             *
       *                                                               *
       *      Error Messages =                                         *
       *            Unexpected SQLCODE from DSN8.DSN8EC1 during        *
       *              <command> request.  <DSNTIAR detail>             *
       *            Unexpected return code from ODBA:                  *
```

```
      *             - Command ............. <command>               *
      *             - AIB return code ...... <AIBRETRN>             *
      *             - AIB reason code ...... <AIBREASN>             *
      *             - DC error call ........ <DC-ERROR-CALL>        *
      *                                                             *
      *   External References =                                     *
      *      Routines/Services =                                    *
      *            DSN8EC1  -     DB2 sample ODBA stored procedure  *
      *            DSNTIAR  -     DB2 SQLCODE message formatter     *
      *                                                             *
      *      Data areas        =  None                              *
      *                                                             *
      *      Control Blocks    =                                    *
      *            SQLCA    -     SQL communication area            *
      *                                                             *
      *   Tables =  None                                            *
      *                                                             *
      *                                                             *
      *   Change Activity = None                                    *
      *                                                             *
      *                                                             *
      *  *Pseudocode*                                               *
      *                                                             *
      *  PROCEDURE A00000-ODBA-SP-CLIENT                            *
      *    Call A30000-ADD-ENTRY to generate add request            *
      *         Call C31000-CALL-ODBA-SP to handle add request      *
      *              Call DSN8.DSN8EC1 to perform add request       *
      *              Call D31100-CHECK-SQLCODE to verify DB2 call   *
      *                   Call E31110-DETAIL-SQL-ERROR to format err *
      *                        Call F31111-PRINT-SQL-ERROR-MSG      *
      *              Call D31200-CHECK-AIBCODE to verify IMS state  *
      *    Call B40000-COMMIT-WORK to commit DB2 work unit          *
      *    Call B50000-DISPLAY-ENTRY to generate display request    *
      *         Call C31000-CALL-ODBA-SP to handle display request  *
      *              Call DSN8.DSN8EC1 to perform display request   *
      *              Call D31100-CHECK-SQLCODE to verify DB2 call   *
      *                   Call E31110-DETAIL-SQL-ERROR to format err *
      *                        Call F31111-PRINT-SQL-ERROR-MSG      *
      *              Call D31200-CHECK-AIBCODE to verify IMS state  *
      *                                                             *
      *-------------------------------------------------------------*



       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370.
       OBJECT-COMPUTER.  IBM-370.

       INPUT-OUTPUT SECTION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.

      *****************************************************************
      * Fields for receiving
      *****************************************************************
       01 DB2-SERVER-LOCATION-NAME PIC X(16).
       01 IMS-SUBSYSTEM-NAME       PIC X(8).

      *****************************************************************
      * Parameter list for invoking sample DB2 stored procedure DSN8EC1
      *****************************************************************
       01 DB2IO-TDBCTLID           PIC X(8).
       01 DB2IO-COMMAND            PIC X(8).
       01 DB2IO-LAST-NAME          PIC X(10).
       01 DB2IO-FIRST-NAME         PIC X(10).
       01 DB2IO-EXTENSION          PIC X(10).
       01 DB2IO-ZIP-CODE           PIC X(7).
       01 DB2OUT-AIBRETRN          PIC S9(9) COMP.
       01 DB2OUT-AIBREASN          PIC S9(9) COMP.
       01 DC-ERROR-CALL            PIC X(4).

      *****************************************************************
      * Buffer for receiving SQL error messages
      *****************************************************************
       01 ERROR-MESSAGE.
          02  ERROR-LEN            PIC S9(4)   COMP VALUE +960.
          02  ERROR-TEXT           PIC X(120)  OCCURS 10 TIMES
                                        INDEXED BY ERROR-INDEX.
       77 ERROR-TEXT-LEN           PIC S9(9)   COMP VALUE +120.

```

```
******************************************************************
* Job status indicator
******************************************************************
 01 RUN-STATUS              PIC X(4).
    88 NOT-OKAY                         VALUE 'BAD'.
    88 OKAY                             VALUE 'GOOD'.

******************************************************************
* Include Cobol standard language global variables
******************************************************************
      EXEC SQL INCLUDE SQLCA END-EXEC.



 LINKAGE SECTION.

******************************************************************
* DSN8EC2 invocation parameter list
******************************************************************
 01 PARMS.
    05 PARMS-LEN            PIC 9(4) USAGE BINARY.
    05 PARMS-DATA           PIC X(25).



 PROCEDURE DIVISION
     USING PARMS.
******************************************************************
* Main driver: Use ODBA to add to and display from the IMS IVP DB
******************************************************************
 A00000-ODBA-SP-CLIENT.
      DISPLAY '****************************************'
              '****************************************'.
      DISPLAY '* DSN8EC2: Sample Client for IMS/ODBA '
              'DB2 stored procedure sample (DSN8.DSN8EC1) '.
      DISPLAY '*'.
      MOVE 'GOOD' TO RUN-STATUS.

      PERFORM B10000-PROCESS-PARMS.

      PERFORM B20000-CONNECT-TO-SERVER.

      IF OKAY THEN
         PERFORM B30000-ADD-ENTRY.

      IF OKAY THEN
         PERFORM B40000-COMMIT-WORK.

      IF OKAY THEN
         PERFORM B50000-DISPLAY-ENTRY.

      DISPLAY '****************************************'
              '****************************************'.

      IF NOT-OKAY THEN
         MOVE 8 to RETURN-CODE.

      STOP RUN.


 B10000-PROCESS-PARMS.
******************************************************************
* Process DSN8EC2 invocation parameters
******************************************************************
      UNSTRING PARMS-DATA
        DELIMITED BY SPACE
        INTO DB2-SERVER-LOCATION-NAME
             IMS-SUBSYSTEM-NAME.
      MOVE IMS-SUBSYSTEM-NAME TO DB2IO-TDBCTLID.


 B20000-CONNECT-TO-SERVER.
******************************************************************
* Connect to the remote server
******************************************************************
      DISPLAY '*    Now connecting to ' DB2-SERVER-LOCATION-NAME.
      DISPLAY '*    for access to IMS node '
                    IMS-SUBSYSTEM-NAME.
      DISPLAY '*'.

      EXEC SQL CONNECT TO :DB2-SERVER-LOCATION-NAME END-EXEC.
      IF SQLCODE IS NOT EQUAL TO ZERO THEN
```

```
         PERFORM D31100-CHECK-SQLCODE.


 B30000-ADD-ENTRY.
******************************************************************
* Generate and add an entry to the IMS IVP database DFSIVD1
******************************************************************
      MOVE 'ADD' TO DB2IO-COMMAND.
      MOVE 'DOE' TO DB2IO-LAST-NAME.
      MOVE 'JOHN' TO DB2IO-FIRST-NAME.
      MOVE '9-876-5432' TO DB2IO-EXTENSION.
      MOVE '98765' TO DB2IO-ZIP-CODE.
      MOVE 0 TO DB2OUT-AIBRETRN.
      MOVE 0 TO DB2OUT-AIBREASN.
      MOVE '    ' TO DC-ERROR-CALL.

      PERFORM C31000-CALL-ODBA-SP.

      IF OKAY THEN
         DISPLAY '*     Entry for:'
         DISPLAY '*      - Last Name .......... ' DB2IO-LAST-NAME
         DISPLAY '*      - First Name ......... ' DB2IO-FIRST-NAME
         DISPLAY '*      - Extension Number ... ' DB2IO-EXTENSION
         DISPLAY '*      - Internal Zip Code .. ' DB2IO-ZIP-CODE
         DISPLAY '*      added successfully to database DFSIVD1.'
         DISPLAY '*'.


 B40000-COMMIT-WORK.
******************************************************************
*  Commit changes in the IMS telephone database
******************************************************************
      EXEC SQL COMMIT
      END-EXEC.

      PERFORM D31100-CHECK-SQLCODE.


 B50000-DISPLAY-ENTRY.
******************************************************************
* Retrieve an entry from IMS IVP database DFSIVD1
******************************************************************
      MOVE 'DIS' TO DB2IO-COMMAND.
      MOVE 'LAST1' TO DB2IO-LAST-NAME.
      MOVE 'NNNN' TO DB2IO-FIRST-NAME.
      MOVE 'N-NNN-NNNN' TO DB2IO-EXTENSION.
      MOVE 'NNNNN' TO DB2IO-ZIP-CODE.
      MOVE 0 TO DB2OUT-AIBRETRN.
      MOVE 0 TO DB2OUT-AIBREASN.
      MOVE '    ' TO DC-ERROR-CALL.

      PERFORM C31000-CALL-ODBA-SP.

      IF OKAY THEN
         DISPLAY '*     Entry for:'
         DISPLAY '*      - Last Name .......... ' DB2IO-LAST-NAME
         DISPLAY '*      - First Name ......... ' DB2IO-FIRST-NAME
         DISPLAY '*      - Extension Number ... ' DB2IO-EXTENSION
         DISPLAY '*      - Internal Zip Code .. ' DB2IO-ZIP-CODE
         DISPLAY '*      retrieved successfully from DFSIVD1.'
         DISPLAY '*'.


 C31000-CALL-ODBA-SP.
******************************************************************
* Invoke the sample stored procedure for IMS/ODBA
******************************************************************
      EXEC SQL CALL DSN8.DSN8EC1 (:DB2IO-TDBCTLID,
                                  :DB2IO-COMMAND,
                                  :DB2IO-LAST-NAME,
                                  :DB2IO-FIRST-NAME,
                                  :DB2IO-EXTENSION,
                                  :DB2IO-ZIP-CODE,
                                  :DB2OUT-AIBRETRN,
                                  :DB2OUT-AIBREASN,
                                  :DC-ERROR-CALL)
      END-EXEC.

      PERFORM D31100-CHECK-SQLCODE.

      IF OKAY THEN
        PERFORM D31200-CHECK-AIBCODE.
```

```


       D31100-CHECK-SQLCODE.
      ****************************************************************
      * Verify that the prior SQL call completed successfully
      ****************************************************************
            IF SQLCODE NOT = 0 THEN
               MOVE 'BAD' TO RUN-STATUS
               DISPLAY '*    Unexpected SQLCODE from DSN8.DSN8EC1 '
                            'during ' DB2IO-COMMAND ' request.'
               DISPLAY '*'
               PERFORM E31110-DETAIL-SQL-ERROR.


       E31110-DETAIL-SQL-ERROR.
      ****************************************************************
      * Call DSNTIAR to return a text message for an unexpected
      * SQLCODE.
      ****************************************************************
            CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
            IF RETURN-CODE = ZERO
               PERFORM F31111-PRINT-SQL-ERROR-MSG VARYING ERROR-INDEX
                  FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10.

      *                                           **MESSAGE FORMAT
      *                                           **ROUTINE ERROR
      *                                           **PRINT ERROR MESSAG


       F31111-PRINT-SQL-ERROR-MSG.
      ****************************************************************
      * Print message text
      ****************************************************************
            DISPLAY ERROR-TEXT (ERROR-INDEX).


       D31200-CHECK-AIBCODE.
      ****************************************************************
      * Verify that the IMS operation via ODBA succeeded
      ****************************************************************
            IF DB2OUT-AIBRETRN NOT = 0 OR DB2OUT-AIBREASN NOT = 0 THEN
               MOVE 'BAD' TO RUN-STATUS
               DISPLAY '*    Unexpected return code from ODBA:'
               DISPLAY '*     - Command .............' DB2IO-COMMAND
               DISPLAY '*     - AIB return code ......' DB2OUT-AIBRETRN
               DISPLAY '*     - AIB reason code ......' DB2OUT-AIBREASN
               DISPLAY '*     - DC error call ........' DC-ERROR-CALL
               DISPLAY '*'.
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ES1

Accepts a department number from the caller and returns parameters containing the total earnings (salaries and bonuses) for employees in that department, as well as the number of employees who got a bonus.

```
-- DSN8ES1: SOURCE MODULE FOR THE SAMPLE SQL PROCEDURE
--
--    LICENSED MATERIALS - PROPERTY OF IBM
--    5635-DB2
--    (C) COPYRIGHT 2000, 2006 IBM CORP.  ALL RIGHTS RESERVED.
--
--    STATUS = VERSION 9
--
-- Function: Accepts a department number from the caller and returns
--           parameters containing the total earnings (salaries and
--           bonuses) for employees in that department, as well as the
--           number of employees who got a bonus.
--
--           In addition, DSN8ES1 generates a result set that contains
--           the serial no, first and last name, salary, and bonus for
--           each employee in the department who got a bonus.  The
--           result set also contains a sequence number so that it can
--           be read in the order it was generated.
```

```
--
--    Notes:
--      Dependencies:
--      - Requires DB2 precompiler support for SQL procedures (DSNHPSM)
--      - Requires a global temporary table (created in sample job
--        DSNTEJ63) for returning the result.
--
--      Restrictions:
--
-- Module Type: SQL Procedure
--   Processor: DB2 for OS/390 precompiler and IBM C/C++ for OS/390
--              or a subsequent release
--  Attributes: Re-entrant and re-usable
--
-- Entry Point: DSN8ES1
--     Purpose: See Function, above
--
--  Parameters:
--  -    Input: DEPTNO          CHAR(3)
--  -   Output: DEPTSAL         DECIMAL(15,2)
--              BONUSCNT        INTEGER
--
--  Normal Exit:
--   Error Exit:
--
--
--  External References:
--  - EMP               : DB2 Sample Employee Table
--  - DSN8.DSN8ES1_RS_TBL: Global Temporary Table for result set
--
--  Pseudocode:
--  - Clear any residual from result set table
--  - Open cursor on EMP table for employees in department DEPTNO
--  - While more rows:
--    - Add current employee's salary and bonus to total department
--      earnings
--    - If current employee's bonus is greater than zero
--      - increment the department bonus counter
--      - add the employee's serial, first and last name, salary and
--        bonus to the result set table, using the bonus counter as
--        a result set sequence number
--  - If no errors, open the cursor to the result set
--
--
  CREATE PROCEDURE DSN8.DSN8ES1
              ( IN DEPTNO           CHAR(3),
               OUT DEPTSAL          DECIMAL(15,2),
               OUT BONUSCNT         INT )
   PARAMETER CCSID EBCDIC
            FENCED
        RESULT SET 1
          LANGUAGE SQL
 NOT DETERMINISTIC
 MODIFIES SQL DATA
            COLLID DSN8ES!!
   WLM ENVIRONMENT WLMENV
           ASUTIME NO LIMIT
  COMMIT ON RETURN NO

         P1: BEGIN NOT ATOMIC
           DECLARE EMPLOYEE_NUMBER     CHAR(6)      CCSID EBCDIC;
           DECLARE EMPLOYEE_FIRSTNME   CHAR(12)     CCSID EBCDIC;
           DECLARE EMPLOYEE_LASTNAME   CHAR(15)     CCSID EBCDIC;
           DECLARE EMPLOYEE_SALARY     DECIMAL(9,2)    DEFAULT 0;
           DECLARE EMPLOYEE_BONUS      DECIMAL(9,2)    DEFAULT 0;
           DECLARE TOTAL_SALARY        DECIMAL(15,2)   DEFAULT 0;
           DECLARE BONUS_COUNTER       INT             DEFAULT 0;
           DECLARE END_TABLE           INT             DEFAULT 0;

           -- Cursor for result set of employees who got a bonus
           DECLARE DSN8ES1_RS_CSR CURSOR WITH RETURN WITH HOLD FOR
                              SELECT RS_SEQUENCE,
                                     RS_EMPNO,
                                     RS_FIRSTNME,
                                     RS_LASTNAME,
                                     RS_SALARY,
                                     RS_BONUS
                                FROM DSN8.DSN8ES1_RS_TBL
                            ORDER BY RS_SEQUENCE;

           -- Cursor to fetch department employees
           DECLARE C1 CURSOR FOR
```

```
               SELECT EMPNO,
                      FIRSTNME,
                      LASTNAME,
                      SALARY,
                      BONUS
                 FROM EMP
                WHERE WORKDEPT = DEPTNO;

    DECLARE CONTINUE HANDLER FOR NOT FOUND
            SET END_TABLE = 1;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
            SET DEPTSAL = NULL;

    -- Clean residual from the result set table
     DELETE FROM DSN8.DSN8ES1_RS_TBL;

       OPEN C1;

      FETCH C1
       INTO EMPLOYEE_NUMBER,
            EMPLOYEE_FIRSTNME,
            EMPLOYEE_LASTNAME,
            EMPLOYEE_SALARY,
            EMPLOYEE_BONUS;

    -- Process each employee in the department
      WHILE END_TABLE = 0 DO
        -- Update department total salary
        SET TOTAL_SALARY = TOTAL_SALARY
                         + EMPLOYEE_SALARY
                         + EMPLOYEE_BONUS;

        -- If the current employee received a bonus
        IF EMPLOYEE_BONUS > 0.00 THEN
          -- Update department bonus count
          SET BONUS_COUNTER = BONUS_COUNTER + 1;

          -- Add the employee's data to the result set
          INSERT INTO  DSN8.DSN8ES1_RS_TBL
                     ( RS_SEQUENCE,
                       RS_EMPNO,
                       RS_FIRSTNME,
                       RS_LASTNAME,
                       RS_SALARY,
                       RS_BONUS )
               VALUES( P1.BONUS_COUNTER,
                       P1.EMPLOYEE_NUMBER,
                       P1.EMPLOYEE_FIRSTNME,
                       P1.EMPLOYEE_LASTNAME,
                       P1.EMPLOYEE_SALARY,
                       P1.EMPLOYEE_BONUS );
        END IF;

      FETCH C1
       INTO EMPLOYEE_NUMBER,
            EMPLOYEE_FIRSTNME,
            EMPLOYEE_LASTNAME,
            EMPLOYEE_SALARY,
            EMPLOYEE_BONUS;

      END WHILE;

     CLOSE C1;
     -- Set return parameters
     SET DEPTSAL = TOTAL_SALARY;
     SET BONUSCNT = BONUS_COUNTER;

     -- Open the cursor to the result set
     OPEN DSN8ES1_RS_CSR;
  END P1
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8ED3

Demonstrates how to call the sample PSM stored procedure DSN8ES1 using static SQL.

```
/*********************************************************************
* Module name = DSN8ED3 (DB2 sample program)                         *
*                                                                    *
* DESCRIPTIVE NAME = Client for sample PSM Stored Procedure DSN8ES1  *
*                                                                    *
*    LICENSED MATERIALS - PROPERTY OF IBM                            *
*    5675-DB2                                                        *
*    (C) COPYRIGHT 2000 IBM CORP.  ALL RIGHTS RESERVED.              *
*                                                                    *
*    STATUS = VERSION 7                                              *
*                                                                    *
* Function: Demonstrates how to call the sample PSM stored procedure *
*           DSN8ES1 using static SQL.                                *
*                                                                    *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   *
* Module size: See linkedit output                                   *
*  Attributes: Re-entrant and re-usable                              *
*                                                                    *
* Entry Point: DSN8ED3                                               *
*     Purpose: See Function                                          *
*     Linkage: DB2SQL                                                *
*              Invoked via SQL UDF call                              *
*                                                                    *
*                                                                    *
*  Parameters: DSN8ED3 uses the C "main" argument convention of      *
*              argv (argument vector) and argc (argument count).     *
*                                                                    *
*              - ARGV[0]  = (input) pointer to a char[9], null-      *
*                           terminated string having the name of     *
*                           this program (DSN8ED3)                   *
*              - ARGV[1]  = (input) pointer to a char[4], null-      *
*                           terminated string having the department  *
*                           number to be passed to DSN8ES1.          *
*              - ARGV[2]  = (input) pointer to a char[16], null-     *
*                           terminated string having the location    *
*                           name of a server to connect to process   *
*                           the current request.  This parameter is  *
*                           optional.  In its absence, the current   *
*                           location is used.                        *
*                                                                    *
* Normal Exit: Return Code: 0000                                     *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code: 0008                                     *
*              - Message: DSN8ED3 failed: Invalid parameter count    *
*                                                                    *
*              - Message: <formatted SQL text from DSNTIAR>          *
*                                                                    *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: DSNTIAR: DB2 msg text formatter   *
*             - Data areas       : None                              *
*             - Control blocks   : None                              *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8ED3:                                                         *
*   - Verify that number of input parameters passed is either:       *
*     - 2 (program name and department number); or                   *
*     - 3 (program name, department number, and (remote) server name *
*     - Other: issue diagnostic message and end with code 0008       *
*   - Connect to server location, if one was passed in.              *
*   - Call sample stored procedure DSN8ES1, passing the department   *
*     number as the argument of the first (input) parameter.         *
*     - if unsuccessful, call sql_error to issue a diagnostic mes-   *
*       sage, then end with code 0008.                               *
*   - Report the following parameters, passed back from DSN8ES1:     *
*     - Total of salary and bonusses for department members          *
```

```
*      - Number of employees in the department who received a bonus   *
*    - If a result set was returned, call processResultSet to handle  *
*      it                                                             *
*    End DSN8ED3                                                      *
*                                                                     *
*    processResultSet:                                                *
*    - Associate a locator with the result set passed from DSN8ES1,   *
*      which contains the serial number, first and last name, salary, *
*      and bonus for each department member who got a bonus.          *
*      - if unsuccessful, call sql_error to issue a diagnostic mes-   *
*        sage, then end with code 0008.                               *
*    - Allocate DSN8ES1_RS_CSR as a cursor for the locator            *
*      - if unsuccessful, call sql_error to issue a diagnostic mes-   *
*        sage, then end with code 0008.                               *
*    - Do while not end of cursor                                     *
*      - Read the cursor                                              *
*      - If successful, print the row as a report line item           *
*      - else if not end of cursor, call sql_error to issue a diag-   *
*        nostic message, then end with code 0008.                     *
*    - Close the cursor                                               *
*      - if unsuccessful, call sql_error to issue a diagnostic mes-   *
*        sage, then end with code 0008.                               *
*    End processResultSet                                             *
*                                                                     *
*    sql_error:                                                       *
*    - call DSNTIAR to format the unexpected SQLCODE.                 *
*    End sql_error                                                    *
*                                                                     *
*********************************************************************/
/********************** C library definitions ***********************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <decimal.h>

/***************************** Equates ******************************/
#define         NULLCHAR     '\0'     /* Null character            */

#define         OUTLEN        80      /* Length of output line      */
#define         DATA_DIM      10      /* Number of message lines    */

#define         NOT_OK         0      /* Run status indicator: Error*/
#define         OK             1      /* Run status indicator: Good */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
  char          hvDeptNo[4];          /* ID of department to query  */
  short int     niDeptNo        = 0;  /* Indic var for dept number  */

  char          hvServerName[17];     /* Location name of server    */

  decimal(15,2) hvDeptEarnings  = 0;  /* Total dept salaries & bonus*/
  short int     niDeptEarnings  = 0;  /* Indic var for dept salary  */

  long int      hvDeptBonusCount= 0;  /* Total no. of bonuses in dpt*/
  short int     niDeptBonusCount= 0;  /* Indic var for dpt bonus cnt*/

  long int      hvSequence;           /* Result set row sequence no.*/
  char          hvEmpno[7];           /* Employee number            */
  char          hvFirstName[13];      /* Employee first name        */
  char          hvLastName[16];       /* Employee last name         */
  decimal(9,2)  hvSalary        = 0;  /* Employee salary            */
  decimal(9,2)  hvBonus         = 0;  /* Employee bonus             */

EXEC SQL END DECLARE SECTION;


/********************* DB2 Result Set Locators **********************/
EXEC SQL BEGIN DECLARE SECTION;
  static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ES1_rs_loc;
EXEC SQL END DECLARE SECTION;


/********************** DB2 Message Formatter ***********************/
struct          error_struct          /* DSNTIAR message structure  */
{
  short int     error_len;
```

```
  char          error_text[DATA_DIM][OUTLEN];
}               error_message = {DATA_DIM * (OUTLEN)};

#pragma         linkage( dsntiar, OS )

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );


/******************** DSN8ED3 Global Variables ********************/
short int       status = OK;          /* DSN8ED3 run status         */

long int        completion_code = 0;  /* DSN8ED3 return code        */


/******************* DSN8ED3 Function Prototypes ******************/
int main( int argc, char *argv[] );
void processResultSet( void );
void sql_error( char locmsg[] );


int main( int argc, char *argv[] )
/******************************************************************
* Get input parms, pass them to DSN8ES1, and process the results     *
******************************************************************/
{
  printf( "**** DSN8ED3: Sample client for DB2 PSM "
          "Stored Procedure Sample (DSN8ES1)\n\n" );

  if( argc == 2 )                     /* Only dept no. was passed   */
    {
      strcpy( hvDeptNo,argv[1] );
    }
  else if( argc == 3 )                /* Dept & server name passed  */
    {
      strcpy( hvDeptNo,argv[1] );
      strcpy( hvServerName,argv[2] );
      EXEC SQL CONNECT TO :hvServerName;
      if( SQLCODE != 0 )
        sql_error( " *** Connect to server" );
    }
  else
    {
      printf( "DSN8ED3 failed: Invalid parameter count\n" );
      status = NOT_OK;
    }

  if( status == OK )
    printf( "Salary and Bonus Report for Department %s\n",hvDeptNo );

  if( status == OK )
    {
      EXEC SQL CALL DSN8.DSN8ES1( :hvDeptNo        :niDeptNo,
                                  :hvDeptEarnings  :niDeptEarnings,
                                  :hvDeptBonusCount:niDeptBonusCount );
      if( SQLCODE != 0 && SQLCODE != 466 )
        sql_error( " *** Call DSN8ES1" );
      else
        {
          printf( "Total Department Salaries and Bonuses: %D(15,2)\n",
                  hvDeptEarnings );
          printf( "Total Number of Bonuses in Department: %i\n",
                  hvDeptBonusCount );
        }
    }

  if( SQLCODE == 0  &&  status == OK )
    if( hvDeptBonusCount != 0 )
      {
        printf( "\n*** Error: Result set was expected from DSN8ES1 "
                       "but was not received\n" );
        status = NOT_OK;
      }

  if( SQLCODE == 466  &&  status == OK )
    processResultSet();

  if( status != OK )
    completion_code = 8;

  return( completion_code );
```

```
} /* end main */


void processResultSet( void )
/*********************************************************************
* If a result was returned by DSN8ES1, this function will process it *
*********************************************************************/
{
  printf( "Bonus Earners are\n" );

  printf( "Serial   First Name     Last Name         "
          "Salary      Bonus\n" );
  printf( "------   ------------   ---------------   "
          "---------   ---------\n" );

  EXEC SQL ASSOCIATE LOCATOR( :DSN8ES1_rs_loc )
                WITH PROCEDURE DSN8.DSN8ES1;
  if( SQLCODE != 0 )
    sql_error( " *** Associate locator DSN8ES1_rs_loc" );

  if( SQLCODE == 0  &&  status == OK )
    {
      EXEC SQL ALLOCATE  DSN8ES1_RS_CSR
             CURSOR FOR
             RESULT SET :DSN8ES1_rs_loc;
      if( SQLCODE != 0 )
        sql_error( " *** Allocate cursor for DSN8ES1 result set" );
    }

  while( SQLCODE == 0  &&  status == OK )
    {
      EXEC SQL FETCH  DSN8ES1_RS_CSR
                INTO :hvSequence,
                     :hvEmpno,
                     :hvFirstName,
                     :hvLastName,
                     :hvSalary,
                     :hvBonus;
      if( SQLCODE == 0 )
        printf( "%s   %s   %s   %9D(9,2)   %9D(9,2)\n",
                hvEmpno, hvFirstName, hvLastName, hvSalary, hvBonus );
      else if( SQLCODE != 100 )
        sql_error( " *** Fetch from DSN8ES1 result set cursor" );
    }

} /* end void processResultSet( void ) */


/*********************************************************************
**********************************************************************
** SQL error handler                                               **
**********************************************************************
*********************************************************************/
void sql_error( char locmsg[] )                               /*proc*/
{


  short int   rc;                     /* DSNTIAR Return code        */
  int         j,k;                    /* Loop control               */
  static int  lrecl = OUTLEN;         /* Width of message lines     */

  /*******************************************************************
  * set status to prevent further processing                         *
  *******************************************************************/
  status = NOT_OK;

  /*******************************************************************
  * print the locator message                                        *
  *******************************************************************/
  printf( " %.80s\n", locmsg );

  /*******************************************************************
  * format and print the SQL message                                 *
  *******************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<DATA_DIM; j++ )
      {
        for( k=0; k<OUTLEN; k++ )
          putchar(error_message.error_text[j][k] );
        putchar('\n');
```

```
        }
    else
      {
        printf( " *** ERROR: DSNTIAR could not format the message\n" );
        printf( " ***        SQLCODE is %d\n",SQLCODE );
        printf( " ***        SQLERRM is \n" );
        for( j=0; j<sqlca.sqlerrml; j++ )
          printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
      }

  } /* end of sql_error */
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ES2

Accepts a bonus base amount (BONUSBAS) to be awarded to employees who are managers.

```
-- DSN8.DSN8ES2: SOURCE MODULE FOR SAMPLE SQL PROCEDURE
--
--    LICENSED MATERIALS - PROPERTY OF IBM
--    5635-DB2
--    (C) COPYRIGHT 2000, 2006 IBM CORP.  ALL RIGHTS RESERVED.
--
--    STATUS = VERSION 9
--
-- Function: Accepts a bonus base amount (BONUSBAS) to be awarded to
--           employees who are managers.  Determines a bonus premium
--           (BONUSPRM) for each manager, according to the number of
--           employees he or she manages.  Updates the BONUS column of
--           the sample EMP table for each manager with the sum of the
--           bonus base and his or her bonus premium.  Returns the
--           total (BONUSTOT) of all bonuses awarded to managers.
--
--           SQLERRCD is null unless an SQL exception occurs, in which
--           case SQLERRCD is set to the current SQLCODE
--
--    Notes:
--      Dependencies:
--      - Requires DB2 precompiler support for SQL procedures (DSNHPSM)
--
--      Restrictions:
--
-- Module Type: SQL Procedure
--   Processor: DB2 for OS/390 precompiler and IBM C/C++ for OS/390
--              or a subsequent release
--  Attributes: Re-entrant and re-usable
--
-- Entry Point: DSN8ES2
--     Purpose: See Function, above
--
--  Parameters:
--  -    Input: BONUSBAS        DECIMAL(15,2)
--  -   Output: BONUSTOT        DECIMAL(15,2)
--              SQLERRCD        INTEGER
--
--  Normal Exit:
--   Error Exit:
--
--
--  External References:
--  - EMP           : DB2 Sample Employee Table
--
--  Pseudocode:
--  - Open a cursor on sample DEPT and EMP tables, that identifies
--    department managers and the number of persons in each department
--
--  - For each manager found:
--    - Determine the bonus premium according to the number of
--      employees managed: $1000 for more than 10, $500 for 6 to 10,
--      $100 for 1 to 5
--    - Update the manager's bonus in the sample EMP table with the
--      sum of the bonus base and the bonus premium
--    - Add the manager's bonus to the total bonuses bucket.
--  - Return total amount of bonuses awarded
```

```
--                                                                      00580000
  CREATE PROCEDURE DSN8.DSN8ES2                                         00590000
             ( IN BONUSBAS         DECIMAL(15,2),                       00600000
               OUT BONUSTOT         DECIMAL(15,2),                      00610000
               OUT SQLERRCD         INTEGER )                           00620000
   PARAMETER CCSID EBCDIC                                               00630000
            FENCED                                                      00640000
       RESULT SETS 0                                                    00650000
          LANGUAGE SQL                                                  00660000
 NOT DETERMINISTIC                                                      00670000
 MODIFIES SQL DATA                                                      00680000
            COLLID DSN8ES!!                                             00700000
   WLM ENVIRONMENT WLMENV                                               00710000
           ASUTIME NO LIMIT                                             00720000
  COMMIT ON RETURN NO                                                   00760000
                                                                        00770000
         P1: BEGIN NOT ATOMIC                                           00780000
           DECLARE MANAGER_ID          CHAR(6)      CCSID EBCDIC;       00790000
           DECLARE NUM_EMPLOYEES       INT             DEFAULT 0;       00800000
           DECLARE BONUSPRM            DECIMAL(15,2)   DEFAULT 0;       00810000
           DECLARE BONUSBKT            DECIMAL(15,2)   DEFAULT 0;       00820000
           DECLARE END_TABLE           INT             DEFAULT 0;       00830000
           DECLARE SQLCODE             INT;                             00831000
                                                                        00840000
           -- Cursor gets id and no. of direct reports for each manager 00850000
           DECLARE C1 CURSOR FOR                                        00860000
                SELECT DEPT.MGRNO,                                      00870000
                       COUNT(DISTINCT EMP.EMPNO)                        00880000
                  FROM DEPT DEPT,                                       00890000
                       EMP  EMP                                         00900000
                 WHERE EMP.WORKDEPT = DEPT.DEPTNO                       00910000
              GROUP BY EMP.WORKDEPT, DEPT.MGRNO;                        00920000
                                                                        00930000
           DECLARE CONTINUE HANDLER FOR NOT FOUND                       00940000
                   SET END_TABLE = 1;                                   00950000
                                                                        00960000
           DECLARE EXIT HANDLER FOR SQLEXCEPTION                        00970000
                   SET SQLERRCD = SQLCODE;                              00980000
                                                                        00990000
               SET BONUSTOT = NULL;                                     01000000
               SET SQLERRCD = NULL;                                     01010000
                                                                        01020000
              OPEN C1;                                                  01030000
                                                                        01040000
             FETCH C1                                                   01050000
              INTO MANAGER_ID,                                          01060000
                   NUM_EMPLOYEES;                                       01070000
                                                                        01080000
             WHILE END_TABLE = 0 DO                                     01090000
               CASE                                                     01100000
                 WHEN( NUM_EMPLOYEES > 10 ) THEN                        01110000
                   SET BONUSPRM = 1000.00;                              01120000
                 WHEN( NUM_EMPLOYEES > 5 )  THEN                        01130000
                   SET BONUSPRM = 500.00;                               01140000
                 WHEN( NUM_EMPLOYEES > 0 )  THEN                        01150000
                   SET BONUSPRM = 100.00;                               01160000
                 ELSE                                                   01170000
                   SET BONUSPRM = 0.00;                                 01180000
               END CASE;                                                01190000
                                                                        01200000
               UPDATE EMP                                               01210000
                  SET BONUS = BONUSBAS + BONUSPRM                       01220000
                WHERE EMPNO = MANAGER_ID;                               01230000
                                                                        01240000
               SET BONUSBKT = BONUSBKT + BONUSBAS + BONUSPRM;           01250000
                                                                        01260000
             FETCH C1                                                   01270000
              INTO MANAGER_ID,                                          01280000
                   NUM_EMPLOYEES;                                       01290000
                                                                        01300000
             END WHILE;                                                 01310000
                                                                        01320000
            CLOSE C1;                                                   01330000
                                                                        01340000
            SET BONUSTOT = BONUSBKT;                                    01350000
                                                                        01360000
         END P1                                                         01370000
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8ED6

Demonstrates how to use WLM_REFRESH, the sample stored procedure for refreshing a WLM environment .

```
/********************************************************************* 00010000
* Module name = DSN8ED6 (DB2 sample program)                          * 00020000
*                                                                     * 00030000
* DESCRIPTIVE NAME = Caller for sample WLM_REFRESH stored procedure  * 00040000
*                                                                     * 00050000
*    LICENSED MATERIALS - PROPERTY OF IBM                             * 00060000
*    5675-DB2                                                         * 00070000
*    (C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED.          * 00090000
*                                                                     * 00120000
*    STATUS = VERSION 7                                               * 00130000
*                                                                     * 00160000
* Function: Demonstrates how to use WLM_REFRESH, the sample stored   * 00220000
*           procedure for refreshing a WLM environment                * 00230000
*                                                                     * 00240000
* Notes:                                                              * 00250000
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       * 00260000
*                                                                     * 00270000
*   Restrictions:                                                     * 00280000
*                                                                     * 00290000
* Module type: C program                                              * 00300000
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                    * 00310000
* Module size: See linkedit output                                    * 00320000
*  Attributes: Re-entrant and re-usable                               * 00330000
*                                                                     * 00340000
* Entry Point: DSN8ED6                                                * 00350000
*     Purpose: See Function                                           * 00360000
*     Linkage: Standard OS/390 linkage                                * 00370000
*                                                                     * 00380000
*                                                                     * 00390000
*  Parameters: DSN8ED6 uses the C "main" argument convention of      * 00400000
*              argv (argument vector) and argc (argument count).     * 00410000
*                                                                     * 00420000
*              - ARGV[0]  = (input) pointer to a char[9], null-      * 00430000
*                           terminated string having the name of     * 00440000
*                           this program (DSN8ED6)                   * 00450000
*              - ARGV[1]  = (input) pointer to a char[32] null-      * 00460000
*                           terminated string having the name of     * 00470000
*                           the WLM environment to be refreshed.     * 00480000
*              - ARGV[2]  = (input) pointer to a char[4], null-      * 00490000
*                           terminated string having the DB2 sub-    * 00500000
*                           system id associated with the WLM        * 00510000
*                           environment to be refreshed              * 00520000
*              - ARGV[3]  = (input) pointer to a char[8], null-      * 00530000
*                           terminated string having the name of a   * 00540000
*                           secondary authorization id that has      * 00550000
*                           access to the resource profile <ssid>.-  * 00560000
*                           WLM_REFRESH.<wlm-environment-name>.      * 00570000
*                           in resource class DSNR.  WLM_REFRESH     * 00580000
*                           requires READ access on that profile     * 00590000
*                           in order to fulfill a refresh request.   * 00600000
*                                                                     * 00610000
* Normal Exit: Return Code: 0000                                      * 00620000
*              - Message: none                                        * 00630000
*                                                                     * 00640000
*  Error Exit: Return Code: 1999                                      * 00650000
*              - Message: Error: Invalid call parameter count        * 00660000
*                                Specify either 2 or 3 call para-    * 00670000
*                                meters for DSN8ED6, as follows:     * 00680000
*                                1. The name of a WLM environment    * 00690000
*                                   to be refreshed (1-32 characters)* 00700000
*                                2. The DB2 subsystem id (1-4 char-  * 00710000
*                                   acters)                          * 00720000
*                                3. A secondary authorization id for * 00730000
*                                   submitting the refresh request   * 00740000
*                                   (Optional.  1-8 characters)      * 00750000
*                                                                     * 00760000
*              - Message: <formatted SQL text from DSNTIAR>          * 00770000
*                                                                     * 00780000
*                                                                     * 00790000
*    External References:                                             * 00800000
*             - Routines/Services: DSNTIAR: DB2 msg text formatter   * 00810000
*             - Data areas       : None                               * 00820000
```

```
*             - Control blocks   : None                               *
*                                                                     *
*  Pseudocode:                                                        *
*   DSN8ED6:                                                          *
*   - Verify that number of input parameters passed is either:        *
*     - 2 (WLM environment name and DB2 ssid); or                     *
*     - 3 (WLM environment name, DB2 ssid, and secondary auth id      *
*     - Other: issue diagnostic message and end with code 1999        *
*   - Set current SQLID to secondary auth id, if one was passed in    *
*   - Call sample stored procedure WLM_REFRESH                        *
*     - if unsuccessful, call sql_error to issue a diagnostic mes-    *
*       sage, then end with code 1999.                                *
*   - Report the following parameters, passed back from WLM_REFRESH: *
*     - Return code                                                   *
*     - Return message                                                *
*     - Set DSN8ED6 return code from WLM_REFRESH return code          *
*   End DSN8ED6                                                       *
*                                                                     *
*   sql_error:                                                        *
*   - call DSNTIAR to format the unexpected SQLCODE.                  *
*   End sql_error                                                     *
*                                                                     *
* Change log:                                                         *
* 01/15/04 PQ79759  - Increase authID to 9 bytes                      *
*                                                                     *
*********************************************************************/
/********************** C library definitions ***********************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/****************************** Equates *****************************/
#define         OUTLEN        80      /* Length of output line      */
#define         DATA_DIM      10      /* Number of message lines    */

#define         NOT_OK         0      /* Run status indicator: Error*/
#define         OK             1      /* Run status indicator: Good */


/******************* DB2 SQL Communication Area *******************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
  char          wlmEnvName[33];       /* WLM environment name       */
  char          ssID[5];              /* Subsystem name             */

  char          authID[9];            /* Current authorization id   */

  char          message[123];         /* WLM_REFRESH return message */
  long int      code;                 /* WLM_REFRESH return code    */
EXEC SQL END DECLARE SECTION;

/********************* DB2 Message Formatter **********************/
struct          error_struct          /* DSNTIAR message structure  */
{
  short int     error_len;
  char          error_text[DATA_DIM][OUTLEN];
}               error_message = {DATA_DIM * (OUTLEN)};

#pragma         linkage( dsntiar, OS )

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );

/******************** DSN8ED6 Global Variables ********************/
short int       status = OK;          /* DSN8ED6 run status         */
long int        completion_code = 0;  /* DSN8ED6 return code        */

/******************* DSN8ED6 Function Prototypes ******************/
int main( int argc, char *argv[] );
void sql_error( char locmsg[] );


int main( int argc, char *argv[] )
/*********************************************************************
* Get input parms, pass them to DSNTWR, and process the results      *
*********************************************************************/
{ printf( "**** DSN8ED6: Sample caller of WLM_REFRESH stored "
                        "procedure (DSNTWR)\n" );
```

```
  printf( "*\n" );

  if( argc < 3  ||  argc > 6 )
    { printf( "* Error: Invalid call parameter count\n" );
      printf( "*        Specify either 2 or 3 call parameters "
                       "for DSN8ED6, as follows:\n" );
      printf( "*        1. The name of a WLM environment to be "
                         "refreshed (1-32 characters)\n" );
      printf( "*        2. The DB2 subsystem id (1-4 characters)\n" );
      printf( "*        3. A secondary authorization id for "
                         "submitting the refresh request\n" );
      printf( "*           (Optional.  1-8 characters)\n" );
      status = NOT_OK;
    }
  else if( strlen(argv[1]) < 1  ||  strlen(argv[1]) > 32 )
    { printf( "* Error: The WLM environment name must be 1-32 "
                       "characters in length\n" );
      status = NOT_OK;
    }
  else if( strlen(argv[2]) < 1  ||  strlen(argv[2]) > 4 )
    { printf( "* Error: The DB2 subsystem id must be 1-4 "
                       "characters in length\n" );
      status = NOT_OK;
    }
  else if( argc == 4 && (strlen(argv[3]) < 1 || strlen(argv[3]) > 8) )
    { printf( "* Error: The secondary authorization id must be 1-8 "
                       "characters in length\n" );
      status = NOT_OK;
    }
  else
    { strcpy( wlmEnvName,argv[1] );
      strcpy( ssID,argv[2] );
    }

  /*******************************************************************
  * Change authid if one was passed in                               *
  *******************************************************************/
  if( status == OK  &&  argc == 4 )
    { strcpy( authID,argv[3] );

      EXEC SQL SET CURRENT SQLID = :authID;
      if( SQLCODE != 0 )
        sql_error( "Error setting SQLID" );
    }

  /*******************************************************************
  * Call WLM_REFRESH to refresh the specified WLM environment        *
  *******************************************************************/
  if( status == OK )
    { EXEC SQL CALL SYSPROC.WLM_REFRESH( :wlmEnvName,
                                         :ssID,
                                         :message,
                                         :code );
      if( SQLCODE != 0 )
        sql_error( "Error calling SYSPROC.WLM_REFRESH" );
      else
        { printf( "* Results: WLM_REFRESH returned code %i "
                            "and the following message:\n",code );
          printf( "* %s\n",message );
        }
    }

  if( status != OK )
    completion_code = 1999;
  else
    completion_code = code;

  return( completion_code );

} /* end main */


void sql_error( char locmsg[] )       /* SQL message formatter      */
{ short int   rc;                     /* DSNTIAR Return code        */
  int         j,k;                    /* Loop control               */
  static int  lrecl = OUTLEN;         /* Width of message lines     */

  /*******************************************************************
  * set status to prevent further processing                         *
  *******************************************************************/
  status = NOT_OK;

```

```
    /********************************************************************
    * print the locator message                                        *
    ********************************************************************/
    printf( " %.80s\n", locmsg );

    /********************************************************************
    * format and print the SQL message                                 *
    ********************************************************************/
    rc = dsntiar( &sqlca, &error_message, &lrecl );
    if( rc == 0 )
      for( j=0; j<DATA_DIM; j++ )
        {
          for( k=0; k<OUTLEN; k++ )
            putchar(error_message.error_text[j][k] );
          putchar('\n');
        }
    else
      {
        printf( " *** ERROR: DSNTIAR could not format the message\n" );
        printf( " ***        SQLCODE is %d\n",SQLCODE );
        printf( " ***        SQLERRM is \n" );
        for( j=0; j<sqlca.sqlerrml; j++ )
          printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
      }

  } /* end of sql_error */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ED7

Calls Db2-provided stored procedure ADMIN_INFO_SYSPARM, which returns the current settings of the Db2 subsystem parameters.

```
  /*********************************************************************
  * Module name = DSN8ED7 (DB2 sample program)                         *
  *                                                                    *
  * DESCRIPTIVE NAME = Caller for SYSPROC.ADMIN_INFO_SYSPARM           *
  *                    (IFCID 106 formatter stored procedure)          *
  *                                                                    *
  *     Licensed Materials - Property of IBM                           *
  *     5635-DB2                                                       *
  *     (C) COPYRIGHT 1982, 2006 IBM Corp.  All Rights Reserved.       *
  *                                                                    *
  *     STATUS = Version 11                                            *
  *                                                                    *
  * Function: Calls DB2-provided stored procedure ADMIN_INFO_SYSPARM,  *
  *           which returns the current settings of the DB2 subsystem  *
  *           parameters.  These settings are then written in          *
  *           report format to standard output.                        *
  *                                                                    *
  * Notes:                                                             *
  *   Dependencies: Requires IBM C/C++ for z/OS                        *
  *                                                                    *
  *   Restrictions:                                                    *
  *                                                                    *
  * Module type: C program                                             *
  *   Processor: IBM C/C++ for z/OS                                    *
  * Module size: See linkedit output                                   *
  *  Attributes: Re-entrant and re-usable                              *
  *                                                                    *
  * Entry Point: DSN8ED7                                               *
  *     Purpose: See Function                                          *
  *     Linkage: Standard z/OS linkage                                 *
  *                                                                    *
  *                                                                    *
  *  Parameters: none                                                  *
  *                                                                    *
  * Normal Exit: Return Code: 0000                                     *
  *              - Message: report of DB2 subsystem parameter settings *
  *                                                                    *
  *  Error Exit: Return Code: 0012                                     *
  *              - Message: <formatted SQL text from DSNTIAR>          *
  *                                                                    *
  *                                                                    *
```

```
*    External References:                                             *
*             - Routines/Services: DSNTIAR: DB2 msg text formatter    *
*             - Data areas       : None                               *
*             - Control blocks   : None                               *
*                                                                     *
*  Pseudocode:                                                        *
*   DSN8ED7:                                                          *
*   - Call ADMIN_INFO_SYSPARM                                         *
*     - if unsuccessful, call sql_error to issue a diagnostic mes-    *
*       sage, then end with code 0012.                                *
*   - Associate a locator variable with the result set               *
*   - Allocate the result set cursor                                  *
*   - Fetch first row from the result set                             *
*   - Print headings                                                  *
*   - Output the content of the result set's current row and fetch   *
*     the next row, until are rows have been fetched                  *
*   - Check for successful processing of result set                  *
*   End DSN8ED7                                                       *
*                                                                     *
*   sql_error:                                                        *
*   - call DSNTIAR to format the unexpected SQLCODE.                  *
*   End sql_error                                                     *
*                                                                     *
*  Change activity =                                                  *
*    11/07/2012 Convert from SYSPROC.DSNWZP    dn1651_inst1 / dn1651 *
*               to SYSPROC.ADMIN_INFO_SYSPARM                         *
*                                                                     *
*********************************************************************/
/**************************** Equates ******************************/
#define         NOT_OK         0      /* Run status indicator: Error*/
#define         OK             1      /* Run status indicator: Good */

#define         OUTLEN        80      /* Length of DSNTIAR line     */
#define         DATA_DIM      10      /* Number of DSNTIAR lines    */

/**************************** Includes *****************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/******************* DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;

/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;

  /******** Host variables for ADMIN_INFO_SYSPARM parameters ********/
  char          hvDB2_MEMBER[9];      /* host var, target DB2       */
  short int     niDB2_MEMBER    = -1; /* indic var for above parm   */

  long int      hvRETURN_CODE   = 0;  /* host var, return code      */
  short int     niRETURN_CODE   = 0;  /* indic var for above parm   */

  char          hvMSG[1332];          /* host var, status message   */
  short int     niMSG           = 0;  /* indic var for above parm   */

  /****** Result set locator for ADMIN_INFO_SYSPARM result set ******/
  static volatile SQL TYPE IS RESULT_SET_LOCATOR *DB2_SYSPARM_rs_loc;

  /******** Host variables for ADMIN_INFO_SYSPARM result set ********/
  long int      hvROWNUM        = 0;  /* host var, row number       */

  char          hvMACRO[9];           /* host var, zparm macro name */

  char          hvPARAMETER[41];      /* host var, zparm name       */

  char          hvINSTALL_PANEL[9];   /* host var, inst panel name  */
  short int     niINSTALL_PANEL = 0;  /* indic var for above parm   */

  char          hvINSTALL_FIELD[41];  /* host var, inst field name  */
  short int     niINSTALL_FIELD = 0;  /* indic var for above parm   */

  char          hvINSTALL_LOCATION[13];/*host var, inst field numb  */
  short int     niINSTALL_LOCATION = 0;/*indic var for above parm   */

  char          hvVALUE[2049];        /* host var, zparm setting    */

  char          hvADDITIONAL_INFO[201];/*host var, zparm setting    */
  short int     niADDITIONAL_INFO = 0; /*indic var for above parm   */

EXEC SQL END DECLARE SECTION;
```

```
/********************** DB2 Message Formatter **********************/
struct          error_struct         /* DSNTIAR message structure  */
  { short int   error_len;
    char        error_text[DATA_DIM][OUTLEN];
  }             error_message = {DATA_DIM * (OUTLEN)};

#pragma         linkage( dsntiar, OS )

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );

/******************** DSN8ED7 Global Variables ********************/
short int       status = OK;         /* DSN8ED7 run status         */

/******************* DSN8ED7 Function Prototypes *******************/
int main( int argc, char *argv[] );
void sql_error( char locmsg[] );     /* Calls SQL text formatter   */

/*******************************************************************
* Get DB2's current subsystem and DSNHDECP parameter settings
*******************************************************************/
int main( int argc, char *argv[] )
  {
    char          msgBuf[1400];      /* message buffer             */

    /****************************************************************
    * Call SYSPROC.ADMIN_INFO_SYSPARM
    ****************************************************************/
    EXEC SQL
      CALL SYSPROC.ADMIN_INFO_SYSPARM
             ( :hvDB2_MEMBER  :niDB2_MEMBER
             , :hvRETURN_CODE :niRETURN_CODE
             , :hvMSG         :niMSG
             );

    if( SQLCODE != 466 )
      { sprintf( msgBuf
               , "DSN8ED7: Error calling stored procedure "
                          "ADMIN_INFO_SYSPARM: \n"
                 "          Return code=%i,\n"
                 "          Message=%s"
               , hvRETURN_CODE
               , hvMSG
               );
        sql_error( msgBuf );
      }

    /****************************************************************
    * Associate a locator variable with the result set
    ****************************************************************/
    if( status == OK )
      {
        EXEC SQL ASSOCIATE LOCATOR
          (:DB2_SYSPARM_rs_loc)
          WITH PROCEDURE SYSPROC.ADMIN_INFO_SYSPARM;

        if (SQLCODE != 0 )
          { sql_error( "*** Associate result set locator "
                       "call unsuccessful." );
          }
      }

    /****************************************************************
    * Allocate the result set cursor
    ****************************************************************/
    if( status == OK )
      {
        EXEC SQL ALLOCATE DB2_SYSPARM_RS_CSR
          CURSOR FOR
          RESULT SET :DB2_SYSPARM_rs_loc;

        if (SQLCODE != 0 )
          { sql_error( "*** Allocate result set cursor "
                       "call unsuccessful." );
          }
      }

    /****************************************************************
    * Fetch first row from the result set
    ****************************************************************/
    if( status == OK )
```

```
    {
      EXEC SQL FETCH DB2_SYSPARM_RS_CSR
        INTO :hvROWNUM
           , :hvMACRO
           , :hvPARAMETER
           , :hvINSTALL_PANEL    :niINSTALL_PANEL
           , :hvINSTALL_FIELD    :niINSTALL_FIELD
           , :hvINSTALL_LOCATION :niINSTALL_LOCATION
           , :hvVALUE
           , :hvADDITIONAL_INFO
           ;

      if (SQLCODE != 0 )
        { sql_error( "*** Priming fetch of result "
                         "set cursor unsuccessful" );
        }
    }

  /*****************************************************************
  * Write the report header
  *****************************************************************/
  if( status == OK )
    {
      printf( "DSN8ED7: Sample DB2 for z/OS "
              "Configuration Setting Report Generator\n \n" );

      printf( "Macro    Parameter                 "
              "Current                                "
              "Description/                           "
              "Install  Fld \n" );
      printf( "Name     Name                      "
              "Setting                                "
              "Install Field Name                     "
              "Panel ID No. \n" );
      printf( "-------- ------------------------ "
              "-------------------------------------- "
              "-------------------------------------- "
              "-------- ----\n" );
    }

  /*****************************************************************
  * Output the contents of the result set
  *****************************************************************/
  while( SQLCODE == 0 && status == OK )
    {
      if( strcmp( hvMACRO,"DSN6SYSP" ) == 0
       || strcmp( hvMACRO,"DSN6LOGP" ) == 0
       || strcmp( hvMACRO,"DSN6ARVP" ) == 0
       || strcmp( hvMACRO,"DSN6SPRM" ) == 0
       || strcmp( hvMACRO,"DSN6FAC"  ) == 0
       || strcmp( hvMACRO,"DSN6GRP"  ) == 0
       || strcmp( hvMACRO,"DSNHDECP" ) == 0
        )
        printf( "%-9.8s"
                "%-26.25s"
                "%-40.39s"
                "%-40.39s"
                "%-9.8s"
                "%4.4s\n"
              , hvMACRO
              , hvPARAMETER
              , hvVALUE
              , hvINSTALL_FIELD
              , hvINSTALL_PANEL
              , hvINSTALL_LOCATION
              );

      EXEC SQL FETCH DB2_SYSPARM_RS_CSR
        INTO :hvROWNUM
           , :hvMACRO
           , :hvPARAMETER
           , :hvINSTALL_PANEL    :niINSTALL_PANEL
           , :hvINSTALL_FIELD    :niINSTALL_FIELD
           , :hvINSTALL_LOCATION :niINSTALL_LOCATION
           , :hvVALUE
           , :hvADDITIONAL_INFO
           ;
    }

  /*****************************************************************
  * Check for successful processing of result set
  *****************************************************************/
```

```
    if (SQLCODE != 100 && status == OK )
      { sql_error( "*** Fetch of result set cursor "
                   "unsuccessful." );

      }

    if( status == OK )
      return( 0 );
    else
      return( 12 );
  } /* end: main */

void sql_error( char locmsg[] )       /* SQL message formatter      */
  { short int   rc;                   /* DSNTIAR Return code        */
    int         j,k;                  /* Loop control               */
    static int  lrecl = OUTLEN;       /* Width of message lines     */

    /*****************************************************************
    * Set status to prevent further processing
    *****************************************************************/
    status = NOT_OK;

    /*****************************************************************
    * Print the locator message
    *****************************************************************/
    printf( " %s\n", locmsg );

    /*****************************************************************
    * Format and print the SQL message
    *****************************************************************/
    rc = dsntiar( &sqlca, &error_message, &lrecl );
    if( rc == 0 )
      for( j=0; j<DATA_DIM; j++ )
        { for( k=0; k<OUTLEN; k++ )
            putchar(error_message.error_text[j][k] );
          putchar('\n');
        }
    else
      { printf( " *** ERROR: DSNTIAR could not format the message\n" );
        printf( " ***        SQLCODE is %d\n",SQLCODE );
        printf( " ***        SQLERRM is \n" );
        for( j=0; j<sqlca.sqlerrml; j++ )
          printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
      }

  } /* end of sql_error */
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ED9

Demonstrates how to use an application program to call DSN8ES3, a sample native SQL procedure.

```
/********************************************************************* 00010002
* Module name = DSN8ED9 (sample program)                             * 00020002
*                                                                    * 00030002
* DESCRIPTIVE NAME: Sample client for:                               * 00040002
*                   DSN8ES3 (DB2 sample native SQL procedure)        * 00050002
*                                                                    * 00060002
*                                                                    * 00070003
*     LICENSED MATERIALS - PROPERTY OF IBM                           * 00071003
*     5635-DB2                                                       * 00072003
*     (C) COPYRIGHT 2006 IBM CORP.  ALL RIGHTS RESERVED.             * 00073003
*                                                                    * 00074003
*     STATUS = VERSION 9                                             * 00075003
*                                                                    * 00080002
* Function: Demonstrates how to use an application program to call   * 00090002
*           DSN8ES3, a sample native SQL procedure.  DSN8ED9         * 00100002
*           receives the schema and name of a stored procedure       * 00110002
*           and passes it to DSN8ES3 to request the CREATE PROCEDURE * 00120002
*           statement.                                               * 00130002
*                                                                    * 00140002
* Notes:                                                             * 00150002
*   Dependencies: Requires DSN8.DSN8ES3                              * 00160002
*                                                                    * 00170002
```

```
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: DB2 Precompiler                                       *
*              IBM C/C++ for z/OS                                    *
* Module size: See linkedit output                                   *
*  Attributes: Reentrant and reusable                                *
*                                                                    *
* Entry point: DSN8ED9                                               *
*     Purpose: See Function                                          *
*     Linkage: Standard MVS program invocation, three parameters.    *
*                                                                    *
*  Parameters: DSN8ED9 uses the C "main" argument convention of      *
*              argv (argument vector) and argc (argument count).     *
*                                                                    *
*              - ARGV[0]: (input) pointer to a char[9],              *
*                         null-terminated string having the name of  *
*                         this program (DSN8ED9)                     *
*              - ARGV[1]: (input) pointer to a char[129],            *
*                         null-terminated string having the schema   *
*                         of a stored procedure                      *
*              - ARGV[2]: (input) pointer to a char[129],            *
*                         null-terminated string having the name of  *
*                         a stored procedure                         *
*              - ARGV[3]: (input) pointer to a char[17],             *
*                         null-terminated string having the name of  *
*                         the server where DSN8ES3 is to be run.     *
*                         This is an optional parameter; the local   *
*                         server is used if no argument is provided. *
*                                                                    *
*      Inputs: None                                                  *
*                                                                    *
*     Outputs: Standard output (SYSPRINT)                            *
*                                                                    *
* Normal Exit: Return Code: 0                                        *
*              - Message: CREATE PROCEDURE statement for specified   *
*                         stored procedure                           *
*                                                                    *
* Normal with Warnings Exit: Return Code: 0004                       *
*              - Message: DSN8ES3 ran successfully but returned      *
*                         no output                                  *
*                                                                    *
*  Error Exit: Return Code: 0012                                     *
*              - Message: DSN8ES3 has completed with return code <n> *
*              - Message: The length of the argument specified for   *
*                         the <parameter-name> does not fall within  *
*                         the required bounds of <minimum-length>    *
*                         and <maximum-length>                       *
*              - Message: DSN8ED9 was invoked with <parameter-count> *
*                         parameters.  At least 2 parameters are     *
*                         required                                   *
*              - Message: <formatted SQL text from DSNTIAR>          *
*                                                                    *
* External References:                                               *
*              - Routines/Services: DSNTIAR: DB2 msg text formatter  *
*              - Data areas       : None                             *
*              - Control blocks   : None                             *
*                                                                    *
*                                                                    *
* Pseudocode:                                                        *
*   DSN8ED9:                                                         *
*     - call getCallParms to receive and validate call parm arguments*
*     - call connectToLocation                                       *
*     - call callDSN8ES3 to invoke the sample native SQL procedure   *
*     - call processDSN8ES3resultSet to output results from DSN8ES3  *
*   End DSN8ED9                                                      *
*                                                                    *
*                                                                    *
*  Change activity =                                                 *
*                                                                    *
*********************************************************************/

/*********************** C library definitions ***********************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <decimal.h>

/***************************** Equates ******************************/
#define         NULLCHAR     '\0'     /* Null character             */

#define         RETNRM             0  /* Normal  return code     @04*/
```

```
#define         RETWRN            4  /* Warning return code        */
#define         RETERR            8  /* Error return code          */
#define         RETSEV           12  /* Severe error return code   */

enum            flag       {No, Yes}; /* Settings for flags        */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ************************/
EXEC SQL BEGIN DECLARE SECTION;
  long int      hvSequence;          /* Result set row sequence no.*/
  char          hvLine[80];          /* line                       */
  char          hvSpSchema[129];     /* Stored procedure schema    */
  short int     niSpSchema      = 0; /* Indic var for schema       */
  char          hvSpName[129];       /* Stored procedure name      */
  short int     niSpName        = 0; /* Indic var for name         */

  char          hvLocationName[17];  /* Server location name       */

EXEC SQL END DECLARE SECTION;


/********************* DB2 Result Set Locators **********************/
EXEC SQL BEGIN DECLARE SECTION;
  static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ES3_rs_loc;
EXEC SQL END DECLARE SECTION;

/********************* DSN8ED9 Global Variables *********************/
unsigned short  resultSetReturned = 0;/* DSN8ES3 result set status  */
long int        rc              =  0; /* DSN8ED9 return code        */


/******************** DSN8ED9 Function Prototypes *******************/
int main                             /* DSN8ED9 driver             */
  ( int argc,                        /* - Input argument count     */
    char *argv[]                     /* - Input argument vector    */
  );
void getCallParms                    /* Process args to call parms */
  ( int argc,                        /* - Input argument count     */
    char *argv[]                     /* - Input argument vector    */
  );
void connectToLocation( void );      /* Connect to DB2 location    */
void callDSN8ES3( void );            /* Call DSN8ES3               */
void processDSN8ES3resultSet( void ); /* Process DSN8ES3 result set */
void associateResultSetLocator(void); /* Assoc DSN8ES3 RS locator   */
void allocateResultSetCursor( void ); /* Alloc DSN8ES3 RS cursor    */
void writeDSN8ES3results( void );    /* Output DSN8ES3 results     */
void fetchFromResultSetCursor( void );/* Read DSNTSPMP RS cursor    */
void issueInvalidCallParmCountError  /* Handler for parm count err */
  ( int argc                         /* - in: no. parms received   */
  );
void issueInvalidParmLengthError     /* Handler for parm len error */
  ( char *parmName,                  /* - in: identify of parm     */
    int minLength,                   /* - in: min valid length     */
    int maxLength                    /* - in: max valid length     */
  );
void issueSqlError                   /* Handler for SQL error      */
  ( char *locMsg                     /* - in: Call location        */
  );


int main                             /* DSN8ED9 driver             */
  ( int argc,                        /* - Input argument count     */
    char *argv[]                     /* - Input argument vector    */
  )
  /*****************************************************************
  * Get input parms, pass them to DSN8ES3, and process the results  *
  *****************************************************************/
  { printf( "**** DSN8ED9: Sample client for DB2 PSM "
            "Stored Procedure Sample (DSN8ES3)\n\n" );

    /****************************************************************
    * Extract the following information from the call parms:        *
    * (1) The schema of the stored procedure                        *
    * (2) The name of the stored procedure                          *
    * (3) Optional: The name of the location where the stored proc  *
    *               resides                                         *
    ****************************************************************/
    getCallParms( argc,argv );
```

```
    /*************************************************************
    * Connect to location where the stored procedure resides       *
    *************************************************************/
    if( rc < RETSEV  &&  strlen(hvLocationName) > 0 )
      connectToLocation();

    if( rc < RETSEV )
      callDSN8ES3();

    if( rc < RETSEV && resultSetReturned == Yes )
      processDSN8ES3resultSet();

    return( rc );

  } /* end main */


void getCallParms                      /* Process args to call parms */
  ( int argc,                          /* - Input argument count     */
    char *argv[]                       /* - Input argument vector    */
  )
  /***************************************************************
  * Verifies that correct call parms have been passed in:          *
  * - Two parameters (the schema and the name of a stored procedure) *
  *   require an argument                                          *
  * - The third parameter (location name) is optional              *
  ***************************************************************/
  { if( argc < 3  || argc > 4 )
      { issueInvalidCallParmCountError( argc );
      }
    else if( strlen( argv[1] ) < 1  ||  strlen( argv[1] ) > 130 )
      { issueInvalidParmLengthError("Stored procedure schema",
                                    1,130);
      }
    else if( strlen( argv[2] ) < 1  ||  strlen( argv[1] ) > 130 )
      { issueInvalidParmLengthError("Stored procedure name",
                                    1,130);
      }
    else
      { strcpy( hvSpSchema, argv[1] );
        strcpy( hvSpName,   argv[2] );
      }

    if( argc > 3 )
      if( strlen( argv[3] ) < 1  ||  strlen( argv[3] ) > 16 )
        { issueInvalidParmLengthError("Server Location Name",1,16);
        }
      else
        strcpy( hvLocationName,argv[3] );
    else
      hvLocationName[0] = NULLCHAR;

  } /* end of getCallParms */


void connectToLocation( void )         /* Connect to DB2 location    */
  /***************************************************************
  * Connects to the DB2 location specified in call parm number 3   *
  ***************************************************************/
  { EXEC SQL
      CONNECT TO :hvLocationName;

    if( SQLCODE != 0 )
      { issueSqlError( "Connect to location failed" );
      }
  } /* end of connectToLocation */


void callDSN8ES3( void )               /* Run sample native SQL proc */
  /***************************************************************
  * Calls the DSN8ES3 (sample native SQL procedure)                *
  ***************************************************************/
  { printf( "\n");
    printf( "-> Now requesting CREATE PROCEDURE statement for %s.%s\n",
            hvSpSchema, hvSpName );

    EXEC SQL CALL DSN8.DSN8ES3( :hvSpSchema  :niSpSchema,
                                :hvSpName    :niSpName );

    /*************************************************************
    * Analyze status codes from DSN8ES3                            *
```

```
    ****************************************************************/
    if( SQLCODE == 466 )
      { resultSetReturned = Yes;
      }
    else if( SQLCODE == 0 )
      { resultSetReturned = No;
        printf( "\n");
        printf( "-> Call to DSN8ES3 succeeded "
                   "but returned no result\n" );
      }
    else
      { issueSqlError( "Call to DSN8ES3 failed" );
      }

  } /* end of callDSN8ES3 */


void processDSN8ES3resultSet( void )   /* Handle DSN8ES3 result set */
  /******************************************************************
  * Outputs data from the result set returned by DSN8ES3            *
  ******************************************************************/
  {
    /****************************************************************
    * Associate a locator with the result set from DSN8ES3          *
    ****************************************************************/
    associateResultSetLocator();

    /****************************************************************
    * Allocate a cursor for the result set                          *
    ****************************************************************/
    if( rc < RETSEV )
      allocateResultSetCursor();

    /****************************************************************
    * Output data from the result set                               *
    ****************************************************************/
    if( rc < RETSEV )
      writeDSN8ES3results();

  } /* end of processDSN8ES3resultSet */


void associateResultSetLocator(void) /* Associate DSN8ES3 RS locator*/
  /******************************************************************
  * Associates the result set from DSN8ES3 with a result set locator *
  ******************************************************************/
  { EXEC SQL
      ASSOCIATE
        LOCATORS( :DSN8ES3_rs_loc )
      WITH PROCEDURE DSN8.DSN8ES3;

    if( SQLCODE != 0 )
      { issueSqlError( "Associate locator call failed" );
      }

  } /* end of associateResultSetLocator */


void allocateResultSetCursor( void )  /* Alloc DSN8ES3 RS cursor    */
  /******************************************************************
  * Allocates a cursor to the locator for the DSN8ES3 result set     *
  ******************************************************************/
  { EXEC SQL
      ALLOCATE DSN8ES3_RS_CSR
        CURSOR FOR RESULT SET :DSN8ES3_rs_loc;

    if( SQLCODE != 0 )
      { issueSqlError( "Allocate result set cursor call failed" );
      }

  } /* end of allocateResultSetCursor */


void writeDSN8ES3results( void )      /* Print DSN8ES3 results      */
  /******************************************************************
  * Outputs the results returned in the result set from DSN8ES3      *
  ******************************************************************/
  { /****************************************************************
    * Get the first entry in the result set                         *
    ****************************************************************/
    fetchFromResultSetCursor();

```

```
    /*************************************************************
    * Process all rows in the result set                         *
    *************************************************************/
    while( SQLCODE == 0  &&  rc < RETSEV )
      { printf( "%s\n",hvLine );

        if( rc < RETSEV )
          { fetchFromResultSetCursor();
          }
      }

  } /* end of writeDSN8ES3results */


void fetchFromResultSetCursor( void ) /* Read DSN8ES3 RS cursor     */
  /***************************************************************
  * Reads the cursor for the DSN8ES3 result set                   *
  ***************************************************************/
  { memset( hvLine,' ',256 );

    EXEC SQL
      FETCH DSN8ES3_RS_CSR
       INTO :hvSequence,
            :hvLine;

    if( SQLCODE != 0  &&  SQLCODE != 100  &&  rc < RETSEV )
      { issueSqlError( "*** Fetch from result set cursor failed" );
      }
  } /* end of fetchFromResultSetCursor */


void issueInvalidCallParmCountError   /* Handler for parm count err */
  ( int argc                          /* - in: no. parms received   */
  )
  /***************************************************************
  * Called when this program is invoked with an inappropriate number *
  * of call parms.                                                   *
  ***************************************************************/
  { printf( "ERROR: DSN8ED9 was invoked with %i parameters\n",--argc );
    printf( "       - The first two parms (schema and name "
                     "of a stored procedure) are required\n" );
    printf( "       - The third parm (location name) "
                     "is optional\n" );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidCallParmCountError */


void issueInvalidParmLengthError      /* Handler for parm len error */
  ( char *parmName,                   /* - in: identify of parm     */
    int minLength,                    /* - in: min valid length     */
    int maxLength                     /* - in: max valid length     */
  )
  /***************************************************************
  * Called when the length of an argument specified for a DSN8ES3    *
  * parameter (parmName) does not fall within the valid bounds for   *
  * size (minLength and maxLength) for that parameter                *
  ***************************************************************/
  { printf( "ERROR: The length of the argument specified for the %s "
                   "parameter\n",parmName );
    printf( "       does not fall within the required bounds of %i "
                   "and %i\n",minLength,maxLength );
    printf( "-----> Processing halted\n" );
    rc = RETSEV;
  } /* end of issueInvalidParmLengthError */


#pragma linkage(dsntiar, OS)
void issueSqlError                    /* Handler for SQL error      */
  ( char *locMsg                      /* - in: Call location        */
  )
  /***************************************************************
  * Called when an unexpected SQLCODE is returned from a DB2 call    *
  ***************************************************************/
  { struct      error_struct {        /* DSNTIAR message structure  */
      short int   error_len;
      char        error_text[10][80];
      }         error_message = {10 * 80};

    extern short int dsntiar( struct      sqlca         *sqlca,
                              struct      error_struct  *msg,
                              int                       *len );
```

```

    short int   DSNTIARrc;            /* DSNTIAR Return code        */
    int         j;                    /* Loop control               */
    static int  lrecl = 80;           /* Width of message lines     */

    /****************************************************************
    * print the locator message                                     *
    ****************************************************************/
    printf( "ERROR: %-80s\n", locMsg );
    printf( "-----> Processing halted\n" );

    /****************************************************************
    * format and print the SQL message                              *
    ****************************************************************/
    DSNTIARrc = dsntiar( &sqlca, &error_message, &lrecl );
    if( DSNTIARrc == 0 )
      for( j = 0; j <= 10; j++ )
        printf( " %.80s\n", error_message.error_text[j] );
    else
      {
        printf( " *** ERROR: DSNTIAR could not format the message\n" );
        printf( " ***        SQLCODE is %d\n",SQLCODE );
        printf( " ***        SQLERRM is \n" );
        for( j=0; j<sqlca.sqlerrml; j++ )
          printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
      }

    /****************************************************************
    * set severe error code                                         *
    ****************************************************************/
    rc = RETSEV;

  } /* end of issueSqlError */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8ES3

Accepts the schema and name of an external stored procedure and returns a result set that contains the CREATE PROCEDURE statement.

```
-- DSN8ES3: SOURCE MODULE FOR THE SAMPLE NATIVE SQL PROCEDURE
--
--    LICENSED MATERIALS - PROPERTY OF IBM
--    5635-DB2
--    (C) COPYRIGHT 2006 IBM CORP.  ALL RIGHTS RESERVED.
--
--    STATUS = VERSION 9
--
-- Function: Accepts the schema and name of an external stored
--           procedure and returns a result set that contains the
--           CREATE PROCEDURE statement.
--
--    Notes:
--      Dependencies:
--      - Requires support for native SQL procedures
--      - Requires a global temporary table (created in sample job
--        DSNTEJ66) for returning the result.
--
--      Restrictions:
--
-- Module Type: SQL Procedure
--   Processor: DB2 for z/OS Version 9
--              or a subsequent release
--
-- Entry Point: DSN8ES3
--     Purpose: See Function, above
--
--  Parameters:
--  -    Input: spSCHEMA        VARCHAR(128)
--              spNAME          VARCHAR(128)
--  -   Output: (None)
--
--  Normal Exit:
--   Error Exit:
```

```
--
--
--  External References:
--  - SYSIBM.SYSROUTINES : DB2 catalog table for routines
--  - SYSIBM.SYSPARMS    : DB2 catalog table for routine parameters
--  - DSN8.DSN8ES3_RS_TBL: Global Temporary Table for result set
--
--  Pseudocode:
--  - Clear any residual from result set table
--  - Get the stored proc properties from SYSIBM.SYSROUTINES
--    - If not found, return SQLSTATE 38602 and the message:
--      'Requested object not found'
--    - If not a stored proc, return SQLSTATE 38603 and the message:
--      'Object is not a stored procedure'
--    - If not an external stored proc, return SQLSTATE 38604 and the
--      message: 'Object is not an external stored procedure'
--  - Open a cursor on the SYSPARMS table
--  - Fetch the first row
--  - If a row is found, insert the CREATE PROCEDURE clause in the
--    result set
--  - For each row in the SYSPARMS cursor, build a parameter clause:
--    - Start with the parameter type (IN, OUT, or INOUT)
--    - Append the parameter name
--    - Append the parameter data type
--    - For string data types, add the CCSID clause
--    - Insert the entry in the result set table
--  - Build the remaining clauses and insert each in the result set
--    - Build and insert the RESULTS SETS clause
--    - Build and insert the EXTERNAL NAME clause
--    - Build and insert the LANGUAGE clause
--    - Build and insert the SQL data access type clause
--    - Build and insert the PARAMETER STYLE clause
--    - Build and insert the DETERMINISTIC clause
--    - Build and insert the FENCED clause
--    - Build and insert the COLLID clause
--    - Build and insert the WLM ENVIRONMENT clause
--    - Build and insert the ASUTIME clause
--    - Build and insert the STAY RESIDENT clause
--    - Build and insert the PROGRAM TYPE clause
--    - Build and insert the EXTERNAL SECURITY clause
--    - Build and insert the AFTER FAILURE clause
--    - Build and insert the RUN OPTIONS clause
--    - Build and insert the COMMIT ON RETURN clause
--    - Build and insert the SPECIAL REGISTERS clause
--    - Build and insert the CALLED ON NULL INPUT clause
--  - Open the cursor to the result set
--
--  CHANGE ACTIVITY
-- 10/31/2013 Ignore SYSPARMS rows where ORDINAL = 0            PM98341
--
--
  CREATE PROCEDURE DSN8.DSN8ES3
             ( IN spSCHEMA         VARCHAR(128),
               IN spNAME           VARCHAR(128) )
   PARAMETER CCSID EBCDIC
        RESULT SET 1
 NOT DETERMINISTIC
 MODIFIES SQL DATA
           ASUTIME NO LIMIT
  COMMIT ON RETURN NO

  P1: BEGIN NOT ATOMIC
   DECLARE hvLANGUAGE          VARCHAR(24)  CCSID EBCDIC;
   DECLARE hvCOLLID            VARCHAR(128) CCSID EBCDIC;
   DECLARE hvDETERMINISTIC     VARCHAR(17)  CCSID EBCDIC;
   DECLARE hvNULL_CALL         CHAR(1)      CCSID EBCDIC;
   DECLARE hvPARAMETER_STYLE   VARCHAR(18)  CCSID EBCDIC;
   DECLARE hvFENCED            CHAR(1)      CCSID EBCDIC;
   DECLARE hvASUTIME           INTEGER         DEFAULT 0;
   DECLARE hvCOMMIT_ON_RETURN  VARCHAR(3)   CCSID EBCDIC;
   DECLARE hvEXTERNAL_NAME     VARCHAR(762) CCSID EBCDIC;
   DECLARE hvEXTERNAL_SECURITY VARCHAR(7)   CCSID EBCDIC;
   DECLARE hvMAX_FAILURE       SMALLINT        DEFAULT 0;
   DECLARE hvORIGIN            CHAR(1)      CCSID EBCDIC;
   DECLARE hvPROGRAM_TYPE      VARCHAR(4)   CCSID EBCDIC;
   DECLARE hvRESULT_SETS       SMALLINT        DEFAULT 0;
   DECLARE hvROUTINETYPE       CHAR(1)      CCSID EBCDIC;
   DECLARE hvRUNOPTS           VARCHAR(762) CCSID EBCDIC;
   DECLARE hvSPECIAL_REGS      VARCHAR(25)  CCSID EBCDIC;
   DECLARE hvSQL_DATA_ACCESS   VARCHAR(17)  CCSID EBCDIC;
   DECLARE hvSTAYRESIDENT      VARCHAR(3)   CCSID EBCDIC;
   DECLARE hvWLM_ENVIRONMENT   VARCHAR(54)  CCSID EBCDIC;
```

```

DECLARE hvENCODING_SCHEME   VARCHAR(7)   CCSID EBCDIC;
DECLARE hvLENGTH            INTEGER         DEFAULT 0;
DECLARE hvORDINAL           SMALLINT        DEFAULT 0;
DECLARE hvPARMNAME          VARCHAR(128) CCSID EBCDIC;
DECLARE hvROWTYPE           VARCHAR(6)   CCSID EBCDIC;
DECLARE hvSCALE             SMALLINT        DEFAULT 0;
DECLARE hvSUBTYPE           VARCHAR(15)  CCSID EBCDIC;
DECLARE hvTYPENAME          VARCHAR(128) CCSID EBCDIC;

DECLARE RETURN_POINT           CHAR(4)   CCSID EBCDIC;

DECLARE LINE                VARCHAR(384) CCSID EBCDIC;
DECLARE LINE_LENGTH         INT             DEFAULT 0;
DECLARE END_TABLE           INT             DEFAULT 0;

DECLARE OPERATION           VARCHAR(12)  CCSID EBCDIC;

DECLARE ROW                 CHAR(80)     CCSID EBCDIC;
DECLARE ROW_SEQUENCE        SMALLINT        DEFAULT 1;

-- Cursor for result set (CREATE PROCEDURE statement)
DECLARE DSN8ES3_RS_CSR CURSOR WITH RETURN WITH HOLD FOR
                       SELECT RS_SEQUENCE,
                              RS_LINE
                         FROM DSN8.DSN8ES3_RS_TBL
                     ORDER BY RS_SEQUENCE;

-- Cursor to fetch proc parm properties from SYSIBM.SYSPARMS
DECLARE SYSPARMS_CURSOR CURSOR FOR
  SELECT  PARMNAME
        ,CASE ROWTYPE
            WHEN 'P' THEN 'IN    '
            WHEN 'O' THEN 'OUT   '
            WHEN 'B' THEN 'INOUT '
          END
        ,ORDINAL
        ,TYPENAME
        ,LENGTH
        ,SCALE
        ,CASE SUBTYPE
            WHEN 'B' THEN ' FOR BIT DATA'
            WHEN 'M' THEN ' FOR MIXED DATA'
            WHEN 'S' THEN ' FOR SBCS DATA'
            WHEN ' ' THEN ' '
          END
        ,CASE ENCODING_SCHEME
            WHEN 'A' THEN ' ASCII'
            WHEN 'E' THEN ' EBCDIC'
            WHEN 'U' THEN ' UNICODE'
            WHEN ' ' THEN ' '
          END
   FROM SYSIBM.SYSPARMS
  WHERE SCHEMA       = spSCHEMA
    AND SPECIFICNAME = spNAME
    AND ORDINAL <> 0
    ORDER BY ORDINAL
    FOR FETCH ONLY;

DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET END_TABLE = 1;

DECLARE EXIT HANDLER FOR SQLEXCEPTION
  SIGNAL SQLSTATE '38601'
    SET MESSAGE_TEXT = 'Unexpected SQLCODE '
                    || CHAR(SQLCODE)
                    || ' from '
                    || OPERATION;

-- Clean residual from the result set table
DELETE FROM DSN8.DSN8ES3_RS_TBL;

-- Fetch the stored proc properties from SYSIBM.SYSROUTINES
SET END_TABLE = 0;
SET OPERATION = 'SELECT INTO';
SELECT LANGUAGE
      ,COLLID
      ,CASE DETERMINISTIC
          WHEN 'N' THEN 'NOT DETERMINISTIC'
          WHEN 'Y' THEN 'DETERMINISTIC'
          WHEN ' ' THEN ' '
        END
```

```
       ,NULL_CALL
       ,CASE PARAMETER_STYLE
           WHEN 'D' THEN 'DB2SQL'
           WHEN 'G' THEN 'GENERAL'
           WHEN 'N' THEN 'GENERAL WITH NULLS'
           WHEN 'J' THEN 'JAVA'
           WHEN ' ' THEN ' '
         END
       ,FENCED
       ,CASE SQL_DATA_ACCESS
           WHEN 'C' THEN 'CONTAINS SQL'
           WHEN 'M' THEN 'MODIFIES SQL DATA'
           WHEN 'N' THEN 'NO SQL'
           WHEN 'R' THEN 'READS SQL DATA'
           WHEN ' ' THEN ' '
         END
       ,CASE STAYRESIDENT
           WHEN 'N' THEN 'NO'
           WHEN 'Y' THEN 'YES'
           WHEN ' ' THEN ' '
         END
       ,ASUTIME
       ,WLM_ENVIRONMENT
       ,CASE PROGRAM_TYPE
           WHEN 'M' THEN 'MAIN'
           WHEN 'S' THEN 'SUB'
           WHEN ' ' THEN ' '
         END
       ,CASE EXTERNAL_SECURITY
           WHEN 'D' THEN 'DB2'
           WHEN 'U' THEN 'USER'
           WHEN 'C' THEN 'DEFINER'
           WHEN ' ' THEN ' '
         END
       ,CASE COMMIT_ON_RETURN
           WHEN 'N' THEN 'NO'
           WHEN 'Y' THEN 'YES'
           WHEN ' ' THEN ' '
         END
       ,RESULT_SETS
       ,EXTERNAL_NAME
       ,RUNOPTS
       ,CASE SPECIAL_REGS
           WHEN 'D' THEN 'DEFAULT SPECIAL REGISTERS'
           WHEN 'I' THEN 'INHERIT SPECIAL REGISTERS'
           WHEN ' ' THEN ' '
         END
       ,MAX_FAILURE
   INTO hvLANGUAGE
       ,hvCOLLID
       ,hvDETERMINISTIC
       ,hvNULL_CALL
       ,hvPARAMETER_STYLE
       ,hvFENCED
       ,hvSQL_DATA_ACCESS
       ,hvSTAYRESIDENT
       ,hvASUTIME
       ,hvWLM_ENVIRONMENT
       ,hvPROGRAM_TYPE
       ,hvEXTERNAL_SECURITY
       ,hvCOMMIT_ON_RETURN
       ,hvRESULT_SETS
       ,hvEXTERNAL_NAME
       ,hvRUNOPTS
       ,hvSPECIAL_REGS
       ,hvMAX_FAILURE
  FROM SYSIBM.SYSROUTINES
 WHERE SCHEMA       = spSCHEMA
   AND NAME         = spNAME;


 CASE
   WHEN END_TABLE = 1 THEN
     SIGNAL SQLSTATE '38602'
       SET MESSAGE_TEXT = 'Requested object "'
                       || spSCHEMA
                       || '"."'
                       || spNAME
                       || '" not found';
   WHEN hvROUTINETYPE <> 'P' THEN
     SIGNAL SQLSTATE '38603'
       SET MESSAGE_TEXT = 'Object is not a stored procedure';
```

```
    WHEN hvORIGIN <> 'E' THEN
      SIGNAL SQLSTATE '38604'
      SET MESSAGE_TEXT = 'Object is not an external stored procedure';
    ELSE -- NOOP below provided to satisfy requirement for ELSE clause
      SET ROW_SEQUENCE = ROW_SEQUENCE;
  END CASE;

  SET END_TABLE = 0;
  SET OPERATION = 'OPEN CURSOR';
  OPEN SYSPARMS_CURSOR;

  SET OPERATION = 'FIRST FETCH';
  FETCH SYSPARMS_CURSOR
   INTO hvPARMNAME
       ,hvROWTYPE
       ,hvORDINAL
       ,hvTYPENAME
       ,hvLENGTH
       ,hvSCALE
       ,hvSUBTYPE
       ,hvENCODING_SCHEME;

  -- Output the CREATE PROCEDURE clause
  IF END_TABLE = 0 THEN
    SET LINE = 'CREATE PROCEDURE ' || spSCHEMA || '.' || spNAME;
  SET RETURN_POINT = 'A100';
  GOTO INSERTLINE;
  END IF;

  A100: -- Build and output the parameter list
  SET LINE = '  ( ';
  WHILE END_TABLE = 0 DO
    -- Output the parameter type (IN, OUT, or INOUT)
    SET LINE = LINE
          || hvROWTYPE  || ' '
          || hvPARMNAME || ' '
          || RTRIM(hvTYPENAME);
    CASE
      WHEN hvTYPENAME = 'DECIMAL'
        OR hvTYPENAME = 'DEC'
        OR hvTYPENAME = 'NUMERIC' THEN
          SET LINE = LINE || '(' || VARCHAR(hvLENGTH)
                          || ',' || VARCHAR(hvSCALE) || ')';

      WHEN hvTYPENAME = 'FLOAT' THEN
          SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';

      WHEN hvTYPENAME = 'CHARACTER'
        OR hvTYPENAME = 'CHAR'
        OR hvTYPENAME = 'CHARACTER VARYING'
        OR hvTYPENAME = 'CHAR VARYING'
        OR hvTYPENAME = 'VARCHAR'
        OR hvTYPENAME = 'CHARACTER LARGE OBJECT'
        OR hvTYPENAME = 'CHAR LARGE OBJECT'
        OR hvTYPENAME = 'CLOB'
        OR hvTYPENAME = 'GRAPHIC'
        OR hvTYPENAME = 'VARGRAPHIC'
        OR hvTYPENAME = 'DBCLOB'
        OR hvTYPENAME = 'BINARY LARGE OBJECT'
        OR hvTYPENAME = 'BLOB' THEN
          SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';

      ELSE -- busy statement below required to handle ELSE case
        SET ROW_SEQUENCE = ROW_SEQUENCE;
    END CASE;

    IF hvSUBTYPE <> ' ' THEN
      SET LINE = LINE || hvSUBTYPE;
    END IF;
    IF hvENCODING_SCHEME <> ' ' THEN
      SET LINE = LINE || ' CCSID' || RTRIM(hvENCODING_SCHEME);
    END IF;
    SET RETURN_POINT = 'B100';
    GOTO INSERTLINE;

    B100: -- Fetch the next parameter
    SET OPERATION = 'FETCH';
    FETCH SYSPARMS_CURSOR
     INTO hvPARMNAME
         ,hvROWTYPE
         ,hvORDINAL
         ,hvTYPENAME
```

```
       ,hvLENGTH
       ,hvSCALE
       ,hvSUBTYPE
       ,hvENCODING_SCHEME;

  SET LINE = '   ,';
END WHILE;

SET OPERATION = 'CLOSE CURSOR';
CLOSE SYSPARMS_CURSOR;
-- Close the parameter list
SET LINE = '  )';
SET RETURN_POINT = 'C100';
GOTO INSERTLINE;

C100: -- Build remaining clauses for the CREATE PROCEDURE statement

-- Output the RESULTS SETS clause
IF hvRESULT_SETS > 0 THEN
  SET LINE = 'DYNAMIC RESULT SETS ' || VARCHAR(hvRESULT_SETS);
  SET RETURN_POINT = 'D100';
  GOTO INSERTLINE;
END IF;

D100: -- Output the EXTERNAL NAME clause
SET LINE = 'EXTERNAL NAME ' || RTRIM(hvEXTERNAL_NAME);
SET RETURN_POINT = 'E100';
GOTO INSERTLINE;

E100: -- Output the LANGUAGE clause
SET LINE = 'LANGUAGE ' || RTRIM(hvLANGUAGE);
SET RETURN_POINT = 'F100';
GOTO INSERTLINE;

F100: -- Output the SQL data access type clause
IF hvSQL_DATA_ACCESS <> ' ' THEN
  SET LINE = hvSQL_DATA_ACCESS;
  SET RETURN_POINT = 'G100';
  GOTO INSERTLINE;
END IF;

G100: -- Output the PARAMETER STYLE clause
IF hvPARAMETER_STYLE <> ' ' THEN
  SET LINE = 'PARAMETER STYLE ' || hvPARAMETER_STYLE;
  SET RETURN_POINT = 'H100';
  GOTO INSERTLINE;
END IF;

H100: -- Output the DETERMINISTIC clause
IF hvDETERMINISTIC <> ' ' THEN
  SET LINE = hvDETERMINISTIC;
  SET RETURN_POINT = 'I100';
  GOTO INSERTLINE;
END IF;

I100: -- Output the FENCED clause
IF hvFENCED <> ' ' THEN
  SET LINE = 'FENCED';
  SET RETURN_POINT = 'J100';
  GOTO INSERTLINE;
END IF;

J100: -- Output the COLLID clause
IF hvCOLLID <> ' ' THEN
  SET LINE = 'COLLID ' || RTRIM(hvCOLLID);
ELSE
  SET LINE = 'NO COLLID';
END IF;
SET RETURN_POINT = 'K100';
GOTO INSERTLINE;

K100: -- Output the WLM ENVIRONMENT clause
SET LINE = 'WLM ENVIRONMENT ' || RTRIM(hvWLM_ENVIRONMENT);
SET RETURN_POINT = 'L100';
GOTO INSERTLINE;

L100: -- Output the ASUTIME clause
IF hvASUTIME <> 0 THEN
  SET LINE = 'ASUTIME ' || VARCHAR(hvASUTIME);
ELSE
  SET LINE = 'ASUTIME NO LIMIT';
END IF;
```

```
   SET RETURN_POINT = 'M100';
   GOTO INSERTLINE;

   M100: -- Output the STAY RESIDENT clause
   IF hvSTAYRESIDENT <> ' ' THEN
     SET LINE = 'STAY RESIDENT ' || hvSTAYRESIDENT;
     SET RETURN_POINT = 'N100';
     GOTO INSERTLINE;
   END IF;

   N100: -- Output the PROGRAM TYPE clause
   IF hvPROGRAM_TYPE <> ' ' THEN
     SET LINE = 'PROGRAM TYPE ' || hvPROGRAM_TYPE;
     SET RETURN_POINT = 'O100';
     GOTO INSERTLINE;
   END IF;

   O100: -- Output the EXTERNAL SECURITY clause
   IF hvEXTERNAL_SECURITY <> ' ' THEN
     SET LINE = 'SECURITY ' || hvEXTERNAL_SECURITY;
     SET RETURN_POINT = 'P100';
     GOTO INSERTLINE;
   END IF;

   P100: -- Output the AFTER FAILURE clause
   IF hvMAX_FAILURE = -1 THEN
     SET LINE = 'STOP AFTER SYSTEM DEFAULT FAILURES';
   ELSEIF hvMAX_FAILURE = 0 THEN
     SET LINE = 'CONTINUE AFTER FAILURE';
   ELSE
     SET LINE = 'STOP AFTER ' || VARCHAR(hvMAX_FAILURE) || ' FAILURES';
   END IF;
   SET RETURN_POINT = 'Q100';
   GOTO INSERTLINE;

   Q100: -- Output the RUN OPTIONS clause
   IF hvRUNOPTS <> ' ' THEN
     SET LINE = 'RUN OPTIONS ''' || hvRUNOPTS || '''';
     SET RETURN_POINT = 'R100';
     GOTO INSERTLINE;
   END IF;

   R100: -- Output the COMMIT ON RETURN clause
   IF hvCOMMIT_ON_RETURN <> ' ' THEN
     SET LINE = 'COMMIT ON RETURN ' || hvCOMMIT_ON_RETURN;
     SET RETURN_POINT = 'S100';
     GOTO INSERTLINE;
   END IF;

   S100: -- Output the SPECIAL REGISTERS clause
   IF hvSPECIAL_REGS <> ' ' THEN
     SET LINE = hvSPECIAL_REGS;
     SET RETURN_POINT = 'T100';
     GOTO INSERTLINE;
   END IF;

   T100: -- Output the CALLED ON NULL INPUT clause
   IF hvNULL_CALL = 'Y' THEN
     SET LINE = 'CALLED ON NULL INPUT';
     SET RETURN_POINT = 'U100';
     GOTO INSERTLINE;
   END IF;

   U100: -- Finish up
   GOTO DONE;

  INSERTLINE:
     SET LINE_LENGTH = LENGTH(LINE);
     WHILE LINE_LENGTH > 72 DO
       SET ROW  = SUBSTR(LINE, 1, 72) || REPEAT( ' ', 8 );
       SET LINE = SUBSTR(LINE, 73, LINE_LENGTH-72);
       SET LINE_LENGTH = LENGTH(LINE);

       SET ROW_SEQUENCE = ROW_SEQUENCE + 1;
       INSERT INTO  DSN8.DSN8ES3_RS_TBL
                  ( RS_SEQUENCE,
                    RS_LINE )
            VALUES( P1.ROW_SEQUENCE,
                    P1.ROW );
     END WHILE;

     SET ROW  = SUBSTR( (LINE || REPEAT(' ', 80)), 1, 80);
```

```
    SET ROW_SEQUENCE = ROW_SEQUENCE + 1;
    SET OPERATION = 'INSERT';
    INSERT INTO  DSN8.DSN8ES3_RS_TBL
               ( RS_SEQUENCE,
                 RS_LINE )
         VALUES( P1.ROW_SEQUENCE,
                 P1.ROW );
    CASE RETURN_POINT
      WHEN 'A100' THEN GOTO A100;
      WHEN 'B100' THEN GOTO B100;
      WHEN 'C100' THEN GOTO C100;
      WHEN 'D100' THEN GOTO D100;
      WHEN 'E100' THEN GOTO E100;
      WHEN 'F100' THEN GOTO F100;
      WHEN 'G100' THEN GOTO G100;
      WHEN 'H100' THEN GOTO H100;
      WHEN 'I100' THEN GOTO I100;
      WHEN 'J100' THEN GOTO J100;
      WHEN 'K100' THEN GOTO K100;
      WHEN 'L100' THEN GOTO L100;
      WHEN 'M100' THEN GOTO M100;
      WHEN 'N100' THEN GOTO N100;
      WHEN 'O100' THEN GOTO O100;
      WHEN 'P100' THEN GOTO P100;
      WHEN 'Q100' THEN GOTO Q100;
      WHEN 'R100' THEN GOTO R100;
      WHEN 'S100' THEN GOTO S100;
      WHEN 'T100' THEN GOTO T100;
      WHEN 'U100' THEN GOTO U100;
      ELSE             GOTO DONE;
    END CASE;

DONE:
  -- Open the cursor to the result set
  SET OPERATION = 'RS CURSOR';
  OPEN DSN8ES3_RS_CSR;
END P1
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUAD

Returns the current date in one these 34 formats.

```
/*********************************************************************
* Module name = DSN8DUAD (DB2 sample program)                        *
*                                                                    *
* DESCRIPTIVE NAME = Current date reformatter (UDF)                  *
*                                                                    *
*                                                                    *
*     LICENSED MATERIALS - PROPERTY OF IBM                           *
*     5625-DB2                                                       *
*     (C) COPYRIGHT 1998, 2003 IBM CORP.  ALL RIGHTS RESERVED.       *
*                                                                    *
*     STATUS = VERSION 8                                             *
*                                                                    *
*                                                                    *
* Function: Returns the current date in one these 34 formats:        *
*                                                                    *
*             D MONTH YY  D MONTH YYYY  DD MONTH YY  DD MONTH YYYY   *
*                 D.M.YY  D.M.YYYY      DD.MM.YY     DD.MM.YYYY      *
*                 D-M-YY  D-M-YYYY      DD-MM-YY     DD-MM-YYYY      *
*                 D/M/YY  D/M/YYYY      DD/MM/YY     DD/MM/YYYY      *
*                 M/D/YY  M/D/YYYY      MM/DD/YY     MM/DD/YYYY      *
*                 YY/M/D  YYYY/M/D      YY/MM/DD     YYYY/MM/DD      *
*                 YY.M.D  YYYY.M.D      YY.MM.DD     YYYY.MM.DD      *
*                         YYYY-M-D                   YYYY-MM-DD      *
*                         YYYY-D-XX                  YYYY-DD-XX      *
*                         YYYY-XX-D                  YYYY-XX-DD      *
*                                                                    *
*           where:                                                   *
*                                                                    *
*             D: Suppress leading zero if the day is less than 10    *
*            DD: Retain leading zero if the day is less than 10      *
*             M: Suppress leading zero if the month is less than 10  *
*            MM: Retain leading zero if the month is less than 10    *
```

```
*         MONTH: Use English-language name of month                  *
*            XX: Use a capital Roman numeral for month               *
*            XX: Use a capital Roman numeral for month               *
*            YY: Use non-century year format                         *
*          YYYY: Use century year format                             *
*                                                                    *
*           Example invocation:                                      *
*            EXEC SQL SET :today = ALTDATE( "DD MONTH YY" );         *
*                                                                    *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   *
* Module size: See linkedit output                                   *
*  Attributes: Re-entrant and re-usable                              *
*                                                                    *
* Entry Point: DSN8DUAD                                              *
*     Purpose: See Function                                          *
*     Linkage: DB2SQL                                                *
*              Invoked via SQL UDF call                              *
*                                                                    *
*       Input: Parameters explicitly passed to this function:        *
*              - *format      : pointer to a char[14], null-termi-   *
*                               nated string having the desired      *
*                               format for the current date (see     *
*                               "Function", above, for valid formats)*
*              - *niFormat    : pointer to a short integer having    *
*                               the null indicator variable for      *
*                               *format.                             *
*              - *fnName      : pointer to a char[138], null-termi-  *
*                               nated string having the UDF family   *
*                               name of this function.               *
*              - *specificName: pointer to a char[129], null-termi-  *
*                               nated string having the UDF specific *
*                               name of this function.               *
*                                                                    *
*                                                                    *
*      Output: Parameters explicitly passed by this function:        *
*              - *dateOut     : pointer to a char[18], null-termi-   *
*                               nated string to receive the current  *
*                               date in the formatted indicated by   *
*                               *format.                             *
*              - *niDateOut   : pointer to a short integer to re-    *
*                               ceive the null indicator variable    *
*                               for *dateOut.                        *
*              - *sqlstate    : pointer to a char[06], null-termi-   *
*                               nated string to receive the SQLSTATE.*
*              - *message     : pointer to a char[70], null-termi-   *
*                               nated string to receive a diagnostic *
*                               message if one is generated by this  *
*                               function.                            *
*                                                                    *
* Normal Exit: Return Code: SQLSTATE = 00000                         *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code: SQLSTATE = 38601                         *
*              - Message: DSN8DUAD Error: No output format entered   *
*              - Message: DSN8DUAD Error: Unknown format specified   *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: None                              *
*             - Data areas       : None                              *
*             - Control blocks   : None                              *
*                                                                    *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8DUAD:                                                        *
*   - Verify that a valid format for the current date was received:  *
*     - if *format is blank or niFormat is not 0, no format passed:  *
*       - issue SQLSTATE 38601 and a diagnostic message              *
*   - Verify that a valid format for the current date was received:  *
*   - Call formatDate to convert the current date in the indicated   *
*     format.                                                        *
*   - If no errors, unset null indicators, and return SQLSTATE 00000 *
*     else set null indicator and return null date out.              *
*   End DSN8DUAD                                                     *
*                                                                    *
*   formatDate:                                                      *
*   - Use the date format to generate a specification string for     *
```

```
*     the getDate function                                          *
*   - Call getDate                                                  *
*   - Perform edits on the result as appropriate:                   *
*     - call Remove0 to strip leading zeroes from the day and/or    *
*       month                                                       *
*     - call romanMonth to convert the month to a roman numeral     *
*   - If *format is not one of the 34 supported formats:            *
*     - issue SQLSTATE 38601 and a diagnostic message               *
*   End formatDate                                                  *
*                                                                   *
*   getDate:                                                        *
*   - invoke the time() library function to query calendar time.    *
*   - invoke the localtime() library function to convert and correct *
*     for local time                                                *
*   - invoke the strftime() library function to format the date from *
*     from the time vector according to specification generated by  *
*     the local formatDate() function.                              *
*   End getDate                                                     *
*                                                                   *
*   Remove0:                                                        *
*   - check the passed string for a character zero in the passed    *
*     location.                                                     *
*   - if a zero is found, eliminate it by shifting all bytes to its *
*     right 1 byte leftward.                                        *
*   End Remove0                                                     *
*                                                                   *
*   romanMonth                                                      *
*   - convert the month (01-12) to a roman numeral (I-XII).         *
*   End romanMonth                                                  *
*                                                                   *
* Change Log:                                                       *
* 2002/10/17 PQ66488 Fix date truncation error                   @01*
*                                                                   *
*                                                                   *
********************************************************************/

#pragma linkage(DSN8DUAD,fetchable)

/********************** C library definitions ***********************/
#include <stdio.h>
#include <string.h>
#include <time.h>

/***************************** Equates ******************************/
#define     NULLCHAR     '\0'         /* Null character            */

#define     MATCH         0           /* Comparison status: Equal   */
#define     NOT_OK        0           /* Run status indicator: Error*/
#define     OK            1           /* Run status indicator: Good */


/*********************** DSN8DUAD functions ************************/
void DSN8DUAD                         /* main routine               */
( char       *format,                 /* in: format for dateOut     */
  char       *dateOut,                /* out: formatted current date*/
  short int  *niFormat,               /* in: indic var, format      */
  short int  *niDateOut,              /* out: indic var for dateOut */
  char       *sqlstate,               /* out: SQLSTATE              */
  char       *fnName,                 /* in: family name of function*/
  char       *specificName,           /* in: specific name of func  */
  char       *message                 /* out: diagnostic message    */
);

int formatDate                        /* format the current date    */
( char       *dateOut,                /* out: formatted curr date   */
  char       *message,                /* out: diagnostic message    */
  char       *sqlstate,               /* out: SQLSTATE              */
  char       *format                  /* in: desired format         */
);

void getDate                          /* gets curr date, formatted  */
( char       *dateOut,                /* out: formatted current date*/
  char       *dateFmt                 /* in: desired date format    */
);

void Remove0                          /* remove 0 from indic. byte  */
( char       *string,                 /* in/out: character string   */
  short int  loc                      /* in: loc'n of zero to remove*/
);

char *romanMonth();                   /* get roman# of curr month#  */

```

```

/*********************************************************************/
/************************** main routine *****************************/
/*********************************************************************/
void DSN8DUAD                         /* main routine                */
( char        *format,                /* in: format for dateOut      */
  char        *dateOut,               /* out: formatted current date*/
  short int   *niFormat,              /* in: indic var, format       */
  short int   *niDateOut,             /* out: indic var for dateOut */
  char        *sqlstate,              /* out: SQLSTATE               */
  char        *fnName,                /* in: family name of function*/
  char        *specificName,          /* in: specific name of func  */
  char        *message                /* out: diagnostic message     */
)
/*********************************************************************
*                                                                    *
* Assumptions:                                                       *
* - *format        points to a char[14], null-terminated string      *
* - *dateOut       points to a char[18], null-terminated string      *
* - *niFormat      points to a short integer                         *
* - *niDateOut     points to a short integer                         *
* - *sqlstate      points to a char[06], null-terminated string      *
* - *fnName        points to a char[138], null-terminated string     *
* - *specificName  points to a char[129], null-terminated string     *
* - *message       points to a char[70], null-terminated string      *
*********************************************************************/
{

  /************************ local variables *************************/
  short int   status = OK;            /* DSN8DUAD run status         */


  /*******************************************************************
  * Verify that a format has been passed in                          *
  *******************************************************************/
  if( *niFormat || ( strlen( format ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUAD Error: No output format entered" );
      strcpy( sqlstate, "38601" );
    }

  /*******************************************************************
  * Call formatDate to format the current date according to format   *
  *******************************************************************/
  if( status == OK )
    status = formatDate( dateOut, message, sqlstate, format );


  /*******************************************************************
  * If formatting was successful, clear the message buffer and sql-  *
  * state, and unset the SQL null indicator for dateOut.             *
  *******************************************************************/
  if( status == OK )
    {
      *niDateOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /*******************************************************************
  * If errors occurred, clear the dateOut buffer and set the SQL null*
  * indicator.  A diagnostic message and the SQLSTATE have been set  *
  * where the error was detected.                                    *
  *******************************************************************/
  else
    {
      dateOut[0] = NULLCHAR;
      *niDateOut = -1;
    }

  return;

} /* end of DSN8DUAD */


/*********************************************************************/
/**************************** functions ******************************/
/*********************************************************************/
int formatDate                        /* format the current date     */
( char        *dateOut,               /* out: formatted curr date    */
  char        *message,               /* out: diagnostic message     */
```

```
  char        *sqlstate,              /* out: SQLSTATE             */
  char        *format                 /* in: desired format        */
)
/*******************************************************************
* Places the current date formatted according to format in dateOut.  *
* If processing is successful, formatDaete returns OK.  Otherwise, a *
* diagnostic message is placed in message, the SQLSTATE is 38601,    *
* and formatDate returns NOT_OK.                                     *
********************************************************************/
{
  short int   func_status = OK;       /* function status indicator  */
  char        dateFmt[14];            /* date format work buffer    */

  /*****************************************************************
  * get the current date and format it according to format          *
  *****************************************************************/
  if( strcmp( format,"D MONTH YY" ) == MATCH )
    {
      getDate( dateOut,"%d %B %y" );  /* format date as DD MONTH YY */
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"D MONTH YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d %B %Y" );  /* formt date as DD MONTH YYYY*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"DD MONTH YY" ) == MATCH )
    {
      getDate( dateOut,"%d %B %y" );  /* format date as DD MONTH YY */
    }
  else if( strcmp( format,"DD MONTH YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d %B %Y" );  /* formt date as DD MONTH YYYY*/
    }
  else if( strcmp( format,"D.M.YY" ) == MATCH )
    {
      getDate( dateOut,"%d.%m.%y" );  /* format date as DD.MM.YY     */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"D.M.YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d.%m.%Y" );  /* format date as DD.MM.YYYY   */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"DD.MM.YY" ) == MATCH )
    {
      getDate( dateOut,"%d.%m.%y" );  /* format date as DD.MM.YY     */
    }
  else if( strcmp( format,"DD.MM.YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d.%m.%Y" );  /* format date as DD.MM.YYYY   */
    }
  else if( strcmp( format,"D-M-YY" ) == MATCH )
    {
      getDate( dateOut,"%d-%m-%y" );  /* format date as DD-MM-YY     */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"D-M-YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d-%m-%Y" );  /* format date as DD-MM-YYYY   */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"DD-MM-YY" ) == MATCH )
    {
      getDate( dateOut,"%d-%m-%y" );  /* format date as DD-MM-YY     */
    }
  else if( strcmp( format,"DD-MM-YYYY" ) == MATCH )
    {
      getDate( dateOut,"%d-%m-%Y" );  /* format date as DD-MM-YYYY   */
    }
  else if( strcmp( format,"D/M/YY" ) == MATCH )
    {
      getDate( dateOut,"%d/%m/%y" );  /* format date as DD/MM/YY     */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"D/M/YYYY" ) == MATCH )
    {
```

```
      getDate( dateOut,"%d/%m/%Y" );  /* format date as DD/MM/YYYY   */
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
      Remove0( dateOut,0 );           /* strip leading 0 if day < 10*/
     }
    else if( strcmp( format,"DD/MM/YY" ) == MATCH )
     {
      getDate( dateOut,"%d/%m/%y" );  /* format date as DD/MM/YY     */
     }
    else if( strcmp( format,"DD/MM/YYYY" ) == MATCH )
     {
      getDate( dateOut,"%d/%m/%Y" );  /* format date as DD/MM/YYYY   */
     }
    else if( strcmp( format,"M/D/YY" ) == MATCH )
     {
      getDate( dateOut,"%m/%d/%y" );  /* format date as MM/DD/YY     */
      Remove0( dateOut,3 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,0 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"M/D/YYYY" ) == MATCH )
     {
      getDate( dateOut,"%m/%d/%Y" );  /* format date as MM/DD/YYYY   */
      Remove0( dateOut,3 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,0 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"MM/DD/YY" ) == MATCH )
     {
      getDate( dateOut,"%m/%d/%y" );  /* format date as MM/DD/YY     */
     }
    else if( strcmp( format,"MM/DD/YYYY" ) == MATCH )
     {
      getDate( dateOut,"%m/%d/%Y" );  /* format date as MM/DD/YYYY   */
     }
    else if( strcmp( format,"YY/M/D" ) == MATCH )
     {
      getDate( dateOut,"%y/%m/%d" );  /* format date as YY/MM/DD     */
      Remove0( dateOut,6 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"YY/MM/DD" ) == MATCH )
     {
      getDate( dateOut,"%y/%m/%d" );  /* format date as YY/MM/DD     */
     }
    else if( strcmp( format,"YYYY/M/D" ) == MATCH )
     {
      getDate( dateOut,"%Y/%m/%d" );  /* format date as YYYY/MM/DD   */
      Remove0( dateOut,8 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,5 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"YYYY/MM/DD" ) == MATCH )
     {
      getDate( dateOut,"%Y/%m/%d" );  /* format date as YYYY/MM/DD   */
     }
    else if( strcmp( format,"YY.M.D" ) == MATCH )
     {
      getDate( dateOut,"%y.%m.%d" );  /* format date as YY.MM.DD     */
      Remove0( dateOut,6 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,3 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"YY.MM.DD" ) == MATCH )
     {
      getDate( dateOut,"%y.%m.%d" );  /* format date as YY.MM.DD     */
     }
    else if( strcmp( format,"YYYY.M.D" ) == MATCH )
     {
      getDate( dateOut,"%Y.%m.%d" );  /* format date as YYYY.MM.DD   */
      Remove0( dateOut,8 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,5 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"YYYY.MM.DD" ) == MATCH )
     {
      getDate( dateOut,"%Y.%m.%d" );  /* format date as YYYY.MM.DD   */
     }
    else if( strcmp( format,"YYYY-M-D" ) == MATCH )
     {
      getDate( dateOut,"%Y-%m-%d" );  /* format date as YYYY-MM-DD   */
      Remove0( dateOut,8 );           /* strip leading 0 if day < 10*/
      Remove0( dateOut,5 );           /* strip leading 0 if month<10*/
     }
    else if( strcmp( format,"YYYY-MM-DD" ) == MATCH )
     {
      getDate( dateOut,"%Y-%m-%d" );  /* format date as YYYY-MM-DD   */
     }
```

```
  else if( strcmp( format,"YYYY-D-XX" ) == MATCH )
    {
      strcpy( dateFmt, "%Y-%d-" );    /* start format as YYYY-DD-   */
      strcat( dateFmt, romanMonth() );/* append roman# for curr mo. */
      getDate( dateOut,dateFmt );     /* format date as YYYY-DD-XX  */
      Remove0( dateOut,5 );           /* strip leading 0 if day < 10*/
    }
  else if( strcmp( format,"YYYY-DD-XX" ) == MATCH )
    {
      strcpy( dateFmt, "%Y-%d-" );    /* start format as YYYY-DD-   */
      strcat( dateFmt, romanMonth() );/* append roman# for curr mo. */
      getDate( dateOut,dateFmt );     /* format date as YYYY-DD-XX  */
    }
  else if( strcmp( format,"YYYY-XX-D" ) == MATCH )
    {
      strcpy( dateFmt, "%Y-" );       /* start format as YYYY-      */
      strcat( dateFmt, romanMonth() );/* append roman# for curr mo. */
      strcat( dateFmt, "-%d" );       /* append -DD to format       */
      getDate( dateOut,dateFmt );     /* get date as YYYY-XX-DD     */
      Remove0( dateOut,               /* strip leading 0 if day < 10*/
               strlen(dateFmt) );
    }
  else if( strcmp( format,"YYYY-XX-DD" ) == MATCH )
    {
      strcpy( dateFmt, "%Y-" );       /* start format as YYYY-      */
      strcat( dateFmt, romanMonth() );/* append roman# for curr mo. */
      strcat( dateFmt, "-%d" );       /* append -DD to format       */
      getDate( dateOut,dateFmt );     /* get date as YYYY-XX-DD     */
    }
  else
    {
      func_status = NOT_OK;
      strcpy( message,
              "DSN8DUAD Error: Unknown format specified" );
      strcpy( sqlstate, "38601" );
    }

  return( func_status );
} /* end of formatDate */

/*******************************************************************/
/**************************** functions ***************************/
/*******************************************************************/
void getDate                          /* gets curr date, formatted  */
( char        *dateOut,               /* out: formatted current date*/
  char        *dateFmt                /* in: desired date format    */
)
/********************************************************************
* Obtains the current date from the system and formats it according  *
* to the format string in *dateFmt. The result is placed in dateOut.*
*                                                                   *
* This function uses the C function localtime to obtain the system   *
* time and date and the C function strftime to format it according   *
* to *dateFmt.  For this program, the following format tokens were   *
* used.  See the C/C++ library reference manuals for more info.      *
*                                                                   *
* %B = full month name                                              *
* %d = day of the month                                             *
* %m = month (01-12)                                                *
* %y = year with century                                            *
* %Y = year with century                                            *
*******************************************************************/
{
  time_t      t;                      /* buff for system time macro */
  struct tm   *tmPtr;                 /* ->buff for time.h tm struct*/
  short int   i;                      /* len of str rtnd by strftime*/
  char        dateBuff[19];           /* gets formatted date     @01*/

  /*****************************************************************
  * Use the C function localtime to get the current date from the    *
  * system, then use the C function strftime to format it according  *
  * to *dateFmt.                                                     *
  *****************************************************************/
  t = time(NULL);
  tmPtr = localtime(&t);
  i = strftime( dateBuff,
                sizeof(dateBuff)-1,
                dateFmt,
                tmPtr );

  if( i > 0 )
    strcpy( dateOut,dateBuff );
```

```

  return;
} /* end getDate */


void Remove0
( char        *string,                /* in/out: character string   */
  short int   loc                     /* in: loc'n of byte to remove*/
)
/*******************************************************************
* Checks *string at the location indicated by loc and, if a character*
* zero resides there, removes it from *string by shifting all bytes  *
* to the right of it leftward by 1 byte.                             *
*******************************************************************/
{
  short int  i;

  if( string[loc] == '0' )
    {
      for( i=loc; i<(strlen(string)-1); i++ )
        string[i] = string[i+1];
      string[i] = NULLCHAR;
    }

  return;
} /* end Remove0 */


char *romanMonth()
/*******************************************************************
* Returns the roman numeral that corresponds to the month number of  *
* the current month.                                                 *
*******************************************************************/
{
  char        romNum[5];              /* gets roman numeral         */
  char        monthNo[18];            /* gets current month number  */

  /*****************************************************************
  * call getDate to get the month number of the current month        *
  *****************************************************************/
  getDate( monthNo,"%m" );            /* format date as MM          */

  /*****************************************************************
  * look up the roman numeral that corresponds to the current month  *
  *****************************************************************/
  if(      strcmp( monthNo,"01" ) == MATCH ) strcpy( romNum, "I" );
  else if( strcmp( monthNo,"02" ) == MATCH ) strcpy( romNum, "II" );
  else if( strcmp( monthNo,"03" ) == MATCH ) strcpy( romNum, "III" );
  else if( strcmp( monthNo,"04" ) == MATCH ) strcpy( romNum, "IV" );
  else if( strcmp( monthNo,"05" ) == MATCH ) strcpy( romNum, "V" );
  else if( strcmp( monthNo,"06" ) == MATCH ) strcpy( romNum, "VI" );
  else if( strcmp( monthNo,"07" ) == MATCH ) strcpy( romNum, "VII" );
  else if( strcmp( monthNo,"08" ) == MATCH ) strcpy( romNum, "VIII" );
  else if( strcmp( monthNo,"09" ) == MATCH ) strcpy( romNum, "IX" );
  else if( strcmp( monthNo,"10" ) == MATCH ) strcpy( romNum, "X" );
  else if( strcmp( monthNo,"11" ) == MATCH ) strcpy( romNum, "XI" );
  else strcpy( romNum, "XII" );

  /*****************************************************************
  * return the result                                                *
  *****************************************************************/
  return( romNum );

} /* end romanMonth */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUAT

Returns the current time in one these 8 formats.

```
/*******************************************************************
* Module name = DSN8DUAT (DB2 sample program)                        *
*                                                                    *
* DESCRIPTIVE NAME = Current time reformatter (UDF)                  *
*                                                                    *
```

```
*     LICENSED MATERIALS - PROPERTY OF IBM                          *
*     5675-DB2                                                      *
*     (C) COPYRIGHT 1998, 2000 IBM CORP. ALL RIGHTS RESERVED.       *
*                                                                   *
*     STATUS = VERSION 7                                            *
*                                                                   *
* Function: Returns the current time in one these 8 formats:        *
*           formats:                                                *
*                                                                   *
*             H:MM AM/PM   HH:MM AM/PM   HH:MM:SS AM/PM   HH:MM:SS  *
*             H.MM         HH.MM         H.MM.SS          HH.MM.SS  *
*                                                                   *
*           where:                                                  *
*                                                                   *
*             H: Suppress leading zero if the hour is less than 10  *
*            HH: Retain leading zero if the hour is less than 10    *
*             M: Suppress leading zero if the minute is less than 10 *
*            MM: Retain leading zero if the minute is less than 10  *
*         AM/PM: Return time in 12-hour clock format, else 24-hour  *
*                                                                   *
*           Example invocation:                                     *
*            EXEC SQL SET :now = ALTTIME( "HH:MM:SS AM/PM" );       *
*                                                                   *
* Notes:                                                            *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher      *
*                                                                   *
*   Restrictions:                                                   *
*                                                                   *
* Module type: C program                                            *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                  *
* Module size: See linkedit output                                  *
*  Attributes: Re-entrant and re-usable                             *
*                                                                   *
* Entry Point: DSN8DUAT                                             *
*     Purpose: See Function                                         *
*     Linkage: DB2SQL                                               *
*              Invoked via SQL UDF call                             *
*                                                                   *
*       Input: Parameters explicitly passed to this function:       *
*              - *format      : pointer to a char[15], null-termi-  *
*                               nated string having the desired     *
*                               format for the current time (see    *
*                               "Function", above, for valid formats)*
*              - *niFormat    : pointer to a short integer having   *
*                               the null indicator variable for     *
*                               *format.                            *
*              - *fnName      : pointer to a char[138], null-termi- *
*                               nated string having the UDF family  *
*                               name of this function.              *
*              - *specificName: pointer to a char[129], null-termi- *
*                               nated string having the UDF specific *
*                               name of this function.              *
*                                                                   *
*                                                                   *
*      Output: Parameters explicitly passed by this function:       *
*              - *timeOut     : pointer to a char[12], null-termi-  *
*                               nated string to receive the current *
*                               time in the formatted indicated by  *
*                               *format.                            *
*              - *niTimeOut   : pointer to a short integer to re-   *
*                               ceive the null indicator variable   *
*                               for *timeOut.                       *
*              - *sqlstate    : pointer to a char[06], null-termi-  *
*                               nated string to receive the SQLSTATE.*
*              - *message     : pointer to a char[70], null-termi-  *
*                               nated string to receive a diagnostic *
*                               message if one is generated by this *
*                               function.                           *
*                                                                   *
* Normal Exit: Return Code: SQLSTATE = 00000                        *
*              - Message: none                                      *
*                                                                   *
*  Error Exit: Return Code: SQLSTATE = 38601                        *
*              - Message: DSN8DUAT Error: No format entered         *
*              - Message: DSN8DUAT Error: Unknown format specified  *
*                                                                   *
*    External References:                                           *
*             - Routines/Services: None                             *
*             - Data areas       : None                             *
*             - Control blocks   : None                             *
*                                                                   *
*                                                                   *
```

```
*  Pseudocode:                                                        *
*   DSN8DUAT:                                                         *
*   - Verify that a valid format for the current time was received:  *
*     - if *format is blank or niFormat is not 0, no format passed:  *
*       - issue SQLSTATE 38601 and a diagnostic message               *
*   - Call getTime to obtain:                                         *
*     - the current 12-hour clock hour                                *
*     - the current 24-hour clock hour                                *
*     - the current minute                                            *
*     - the current second                                            *
*   - Set AM/PM indicator to PM if 24-hour clock hour > 11, else AM  *
*   - Call remove0prefix to remove leading zeroes from the hour      *
*     component, if appropriate                                       *
*   - Call buildTime to generate the output time using the format to *
*     determine which of the time components, delimiters, and AM/PM  *
*     indicator (if any) to pass.                                     *
*   - If no errors, unset null indicators, and return SQLSTATE 00000 *
*     else set null indicator and return null time out.               *
*   End DSN8DUAT                                                      *
*                                                                     *
*   getTime:                                                          *
*   - invoke the time() library function to query calendar time.     *
*   - invoke the localtime() library function to convert and correct *
*     for local time.                                                 *
*   - invoke the strftime() library function to format 12-hour, 24-  *
*     hour, minute, and second components from the time vector.      *
*   End getTime                                                       *
*                                                                     *
*   buildTime:                                                        *
*   - Concatenate the minutes component to the hours component with  *
*     an intervening delimiter (: or .).                              *
*   - If the seconds component is not null, concatenate it to the    *
*     timestring with an intervening delimiter (: or .)              *
*   - If the AM/PM indicator is not null, concatenate it to the      *
*     timestring                                                      *
*   End buildTime                                                     *
*                                                                     *
*   remove0prefix:                                                    *
*   - eliminate leading zeroes from a hour component                 *
*   End remove0prefix                                                 *
*                                                                     *
*                                                                     *
*********************************************************************/

#pragma linkage(DSN8DUAT,fetchable)

/********************** C library definitions ***********************/
#include <stdio.h>
#include <string.h>
#include <time.h>

/**************************** Equates *****************************/
#define     NULLCHAR     '\0'         /* Null character            */

#define     MATCH          0          /* Comparison status: Equal   */
#define     NOT_OK         0          /* Run status indicator: Error*/
#define     OK             1          /* Run status indicator: Good */


/*********************** DSN8DUAT functions ************************/
void DSN8DUAT                         /* main routine              */
( char        *format,                /* in: format for timeOut     */
  char        *timeOut,               /* out: formatted current time*/
  short int   *niFormat,              /* in: indic var, format      */
  short int   *niTimeOut,             /* out: indic var, timeOut    */
  char        *sqlstate,              /* out: SQLSTATE              */
  char        *fnName,                /* in: family name of function*/
  char        *specificName,          /* in: specific name of func  */
  char        *message                /* out: diagnostic message    */
);

void getTime                          /* Get current time           */
( char        *hh12,                  /* out: hours (12 hour clock) */
  char        *hh24,                  /* out: hours (24 hour clock) */
  char        *mm,                    /* out: minutes               */
  char        *ss                     /* out: seconds               */
);

void buildTime                        /* Format time as specified   */
( char        *timeStr,               /* out: reformatted time      */
  char        *hh,                    /* in: hours component        */
  char        *mm,                    /* in: minutes component      */
```

```
  char        *ss,                    /* in: seconds component      */
  char        *delim,                 /* in: delimiter of choice    */
  char        *suffix                 /* in: AM/PM suffix (if any)  */
);

void remove0prefix                    /* Remove leading zeroes      */
( char        *string                 /* in/out: character string   */
);

/*********************************************************************/
/*************************** main routine ****************************/
/*********************************************************************/
void DSN8DUAT                         /* main routine               */
( char        *format,                /* in: format for timeOut     */
  char        *timeOut,               /* out: formatted current time*/
  short int   *niFormat,              /* in: indic var, format      */
  short int   *niTimeOut,             /* out: indic var, timeOut    */
  char        *sqlstate,              /* out: SQLSTATE              */
  char        *fnName,                /* in: family name of function*/
  char        *specificName,          /* in: specific name of func  */
  char        *message                /* out: diagnostic message    */
)
/*********************************************************************
*                                                                    *
*                                                                    *
* Assumptions:                                                       *
* - *format           points to a char[15], null-terminated string   *
* - *timeOut          points to a char[12], null-terminated string   *
* - *niFormat         points to a short integer                      *
* - *niTimeOut        points to a short integer                      *
* - *sqlstate         points to a char[06], null-terminated string   *
* - *fnName           points to a char[138], null-terminated string  *
* - *specificName     points to a char[129], null-terminated string  *
* - *message          points to a char[70], null-terminated string   *
*********************************************************************/
{
  /************************ local variables ************************/
  short int   status = OK;            /* DSN8DUMN run status        */

  char        hh12[3];                /* current time - hours       */
  char        hh24[3];                /* current time - hours       */
  char        mm[3];                  /* current time - minutes     */
  char        ss[3];                  /* current time - seconds     */
  char        suffix[4];              /* AM/PM indicator            */


  /*******************************************************************
  * Verify that a format has been passed in                          *
  *******************************************************************/
  if( *niFormat || ( strlen( format ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUAT Error: No format entered" );
      strcpy( sqlstate, "38601" );
    }

  /*******************************************************************
  * Get current time in the specified format                         *
  *******************************************************************/
  if( status == OK )
    {
      /***************************************************************
      * Get the current hour (hh), minute (mm), and second (ss)      *
      ***************************************************************/
      getTime( hh12,hh24,mm,ss );

      /***************************************************************
      * Suffix is AM unless it's noon or later on the 24-hour clock  *
      ***************************************************************/
      strcpy( suffix," AM" );
      if( strcmp( hh24,"11" ) > 0 )
        strcpy( suffix," PM" );

      /***************************************************************
      * Format the current time according to the input format        *
      ***************************************************************/
      if( strcmp( format,"H:MM AM/PM" ) == MATCH )
        {
          remove0prefix( hh12 );
          buildTime( timeOut,hh12,mm,"",":",suffix );
        }
```

```
      else if( strcmp( format,"HH:MM AM/PM" ) == MATCH )
        {
          buildTime( timeOut,hh12,mm,"",":",suffix );
        }
      else if( strcmp( format,"HH:MM:SS AM/PM" ) == MATCH )
        {
          buildTime( timeOut,hh12,mm,ss,":",suffix );
        }
      else if( strcmp( format,"HH:MM:SS" ) == MATCH )
        {
          buildTime( timeOut,hh24,mm,ss,":","" );
        }
      else if( strcmp( format,"H.MM" ) == MATCH )
        {
          remove0prefix( hh24 );
          buildTime( timeOut,hh24,mm,"",".","" );
        }
      else if( strcmp( format,"HH.MM" ) == MATCH )
        {
          buildTime( timeOut,hh24,mm,"",".","" );
        }
      else if( strcmp( format,"H.MM.SS" ) == MATCH )
        {
          remove0prefix( hh24 );
          buildTime( timeOut,hh24,mm,ss,".","" );
        }
      else if( strcmp( format,"HH.MM.SS" ) == MATCH )
        {
          buildTime( timeOut,hh24,mm,ss,".","" );
        }
      else
        {
          status = NOT_OK;
          strcpy( message,
                  "DSN8DUAT Error: Unknown format specified" );
          strcpy( sqlstate, "38601" );
        }
    } /* if( status == OK ) */

  /*******************************************************************
  * If operation was successful, clear the message buffer and sql-   *
  * state, and unset the SQL null indicator for timeOut.             *
  *******************************************************************/
  if( status == OK )
    {
      *niTimeOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /*******************************************************************
  * If errors occurred, clear the timeOut buffer and set the SQL null*
  * indicator.  A diagnostic message and the SQLSTATE have been set  *
  * where the error was detected.                                    *
  *******************************************************************/
  else
    {
      timeOut[0] = NULLCHAR;
      *niTimeOut = -1;
    }

  return;

} /* end DSN8DUAT */


/*********************************************************************/
/***************************** functions ****************************/
/*********************************************************************/
void getTime
( char        *hh12,                  /* out: hours (12 hour clock) */
  char        *hh24,                  /* out: hours (24 hour clock) */
  char        *mm,                    /* out: minutes               */
  char        *ss                     /* out: seconds               */
)
/*********************************************************************
* Obtains the current time from the system and returns the hours in  *
* *hh12 (12-hour clock hours) and *hh24 (24-hour clock hours), the   *
* minutes in *mm, and the seconds in *ss.                            *
*********************************************************************/
{
  time_t      t;                      /* buff for system time macro */
  struct tm   *tmptr;                 /* buffer for time.h tm struct*/
```

```
  char        timeBuf[13];            /* buffer to receive sys time */

  short int   i;                      /* len of str rtnd by strftime*/
  char        *tokPtr;                /* string ptr for token parser*/
  char        hhmmss[10];             /* time in HH:MM:SS format    */

  /****************************************************************
  * Use the C function localtime to get the current time from the   *
  * system, then use the C function strftime to format it into a    *
  * string containing both 12- and 24-hour clock hours and minutes  *
  * and seconds, all separated by slashes.                          *
  ****************************************************************/
  t = time(NULL);
  tmptr = localtime(&t);
  i = strftime( timeBuf,
                sizeof(timeBuf)-1,
                "%I/%H/%M/%S",        /* format as hh12/hh24/mm/ss  */
                tmptr );

  /****************************************************************
  * Use the strtok func to extract time components from time buffer  *
  ****************************************************************/
  tokPtr = strtok( timeBuf,"/" );     /* Parse to first slash       */
  strcpy( hh12,tokPtr );              /* for hours on 12-hour clock */

  tokPtr = strtok( NULL,"/" );        /* Parse to 2nd slash         */
  strcpy( hh24,tokPtr );              /* for hours on 24-hour clock */

  tokPtr = strtok( NULL,"/" );        /* Parse to 3rd slash         */
  strcpy( mm,tokPtr );                /* for minutes                */

  tokPtr = strtok( NULL,"/" );        /* Parse remaining bytes      */
  strcpy( ss,tokPtr );                /* for seconds                */

} /* end getTime */


void buildTime
( char        *timeStr,               /* out: reformatted time      */
  char        *hh,                    /* in: hours component        */
  char        *mm,                    /* in: minutes component      */
  char        *ss,                    /* in: seconds component      */
  char        *delim,                 /* in: delimiter of choice    */
  char        *suffix                 /* in: AM/PM suffix (if any)  */
)
/******************************************************************
* Builds *timeStr from hours (*hh), minutes (*mm), and seconds (*ss, *
* if not null), separated by the value in *delim and suffixed by the *
* value, if not null, in *suffix.                                    *
******************************************************************/
{
  /****************************************************************
  * Build timeStr from incoming time components                     *
  ****************************************************************/
  strcpy( timeStr,hh );               /* Start with hours ...       */
  strcat( timeStr,delim );            /* append the delimiter       */
  strcat( timeStr,mm );               /* append minutes             */
  if( strlen(ss) > 0 )                /* and, if seconds specified, */
    {
      strcat( timeStr,delim );        /* ..append the delimiter     */
      strcat( timeStr,ss );           /* ..append seconds           */
    }
  if( strlen(suffix) > 0 )            /* and, if suffix specified,  */
    strcat( timeStr,suffix );         /* ..append it.               */

} /* end buildTime */


void remove0prefix
( char        *string                 /* in/out: character string   */
)
/******************************************************************
* Eliminates all leading zeroes from *string.  Leaves a single zero  *
* in the first byte of *string if *string is all zeroes.             *
******************************************************************/
{
  short int   i = 0;                  /* Loop control               */
  short int   j = 0;                  /* Loop control               */

  /****************************************************************
  * if leading zero in first byte, skip up to first non-zero        *
  ****************************************************************/
```

```
    if( string[0] == '0' )
      for( i=0; string[i] == '0'; i++ );

    /*******************************************************************
    * if at end of string, it was all zeroes: put zero in 1st byte     *
    *******************************************************************/
    if( string[i] == '\0' )
      strcpy( string,"0" );
    /*******************************************************************
    * otherwise, left-shift non-zero chars and terminate string        *
    *******************************************************************/
    else
      {
        for( j=0; string[i] != NULLCHAR; j++ )
          string[j] = string[i++];
        string[j] = NULLCHAR;
      }
  } /* end remove0prefix */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUCD

Converts a given date from one to another of these 34 formats.

```
/*********************************************************************
* Module name = DSN8DUCD (DB2 sample program)                        *
*                                                                    *
* DESCRIPTIVE NAME = General date reformatter (UDF)                  *
*                                                                    *
*                                                                    *
*  LICENSED MATERIALS - PROPERTY OF IBM                              *
*  5675-DB2                                                          *
*  (C) COPYRIGHT 2000 IBM CORP.  ALL RIGHTS RESERVED.                *
*                                                                    *
*  STATUS = VERSION 7                                                *
*                                                                    *
*                                                                    *
* Function: Converts a given date from one to another of these 34    *
*           formats:                                                 *
*                                                                    *
*             D MONTH YY  D MONTH YYYY  DD MONTH YY  DD MONTH YYYY   *
*                 D.M.YY  D.M.YYYY      DD.MM.YY     DD.MM.YYYY      *
*                 D-M-YY  D-M-YYYY      DD-MM-YY     DD-MM-YYYY      *
*                 D/M/YY  D/M/YYYY      DD/MM/YY     DD/MM/YYYY      *
*                 M/D/YY  M/D/YYYY      MM/DD/YY     MM/DD/YYYY      *
*                 YY/M/D  YYYY/M/D      YY/MM/DD     YYYY/MM/DD      *
*                 YY.M.D  YYYY.M.D      YY.MM.DD     YYYY.MM.DD      *
*                         YYYY-M-D                   YYYY-MM-DD      *
*                         YYYY-D-XX                  YYYY-DD-XX      *
*                         YYYY-XX-D                  YYYY-XX-DD      *
*                                                                    *
*           where:                                                   *
*                                                                    *
*             D: Suppress leading zero if the day is less than 10    *
*            DD: Retain leading zero if the day is less than 10      *
*             M: Suppress leading zero if the month is less than 10  *
*            MM: Retain leading zero if the month is less than 10    *
*         MONTH: Use English-language name of month                  *
*            XX: Use a capital Roman numeral for month               *
*            XX: Use a capital Roman numeral for month               *
*            YY: Use non-century year format                         *
*          YYYY: Use century year format                             *
*                                                                    *
*           Example invocation:                                      *
*            EXEC SQL SET :newDate = ALTDATE( "3/15/1947",           *
*                                             "M/D/YYYY",            *
*                                             "DD MONTH YY" );       *
*            ==> newDate = "15 March 47"                             *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   *
```

```
* Module size: See linkedit output                                  * 00600000
*  Attributes: Re-entrant and re-usable                             * 00610000
*                                                                   * 00620000
* Entry Point: DSN8DUCD                                             * 00630000
*     Purpose: See Function                                         * 00640000
*     Linkage: DB2SQL                                               * 00650000
*              Invoked via SQL UDF call                             * 00660000
*                                                                   * 00670000
*       Input: Parameters explicitly passed to this function:       * 00680000
*              - *dateIn      : pointer to a char[18], null-termi-   * 00690000
*                               nated string having a date in the    * 00700000
*                               format indicated by *formatIn.       * 00710000
*              - *formatIn    : pointer to a char[14], null-termi-   * 00720000
*                               nated string having the format of    * 00730000
*                               date found in *dateIn (see "Func-    * 00740000
*                               tion", above, for valid formats).    * 00750000
*              - *formatOut   : pointer to a char[14], null-termi-   * 00760000
*                               nated string having the format to    * 00770000
*                               which the date found in *dateIn is   * 00780000
*                               to be converted.  See "Function",    * 00790000
*                               above, for valid formats.            * 00800000
*              - *niDateIn    : pointer to a short integer having    * 00810000
*                               the null indicator variable for      * 00820000
*                               *dateIn.                             * 00830000
*              - *niFormatIn  : pointer to a short integer having    * 00840000
*                               the null indicator variable for      * 00850000
*                               *formatIn.                           * 00860000
*              - *niFormatOut : pointer to a short integer having    * 00870000
*                               the null indicator variable for      * 00880000
*                               *formatOut.                          * 00890000
*              - *fnName      : pointer to a char[138], null-termi-  * 00900000
*                               nated string having the UDF family   * 00910000
*                               name of this function.               * 00920000
*              - *specificName: pointer to a char[129], null-termi-  * 00930000
*                               nated string having the UDF specific * 00940000
*                               name of this function.               * 00950000
*                                                                   * 00960000
*                                                                   * 00970000
*      Output: Parameters explicitly passed by this function:       * 00980000
*              - *dateOut     : pointer to a char[18], null-termi-   * 00990000
*                               nated string to receive the refor-   * 01000000
*                               matted date.                         * 01010000
*              - *niDateOut   : pointer to a short integer to re-    * 01020000
*                               ceive the null indicator variable    * 01030000
*                               for *dateOut.                        * 01040000
*              - *sqlstate    : pointer to a char[06], null-termi-   * 01050000
*                               nated string to receive the SQLSTATE.* 01060000
*              - *message     : pointer to a char[70], null-termi-   * 01070000
*                               nated string to receive a diagnostic * 01080000
*                               message if one is generated by this  * 01090000
*                               function.                            * 01100000
*                                                                   * 01110000
* Normal Exit: Return Code: SQLSTATE = 00000                        * 01120000
*              - Message: none                                      * 01130000
*                                                                   * 01140000
*  Error Exit: Return Code: SQLSTATE = 38601                        * 01150000
*              - Message: DSN8DUCD Error: No input date entered      * 01160000
*              - Message: DSN8DUCD Error: No input format entered    * 01170000
*              - Message: DSN8DUCD Error: No output format entered   * 01180000
*                                                                   * 01190000
*              Return Code: SQLSTATE = 38602                        * 01200000
*              - Message: DSN8DUCD Error: Unknown input format       * 01210000
*                                         specified                 * 01220000
*              - Message: DSN8DUCD Error: Value for year is incor-   * 01230000
*                                         rect or does not conform   * 01240000
*                                         to input format           * 01250000
*              - Message: DSN8DUCD Error: Value for month is incor-  * 01260000
*                                         rect or does not conform   * 01270000
*                                         to input format           * 01280000
*              - Message: DSN8DUCD Error: Value for day is incor-    * 01290000
*                                         rect or does not conform   * 01300000
*                                         to input format           * 01310000
*                                                                   * 01320000
*              Return Code: SQLSTATE = 38602                        * 01330000
*              - Message: DSN8DUCD Error: Unknown output format      * 01340000
*                                         specified                 * 01350000
*                                                                   * 01370000
*    External References:                                           * 01380000
*             - Routines/Services: None                             * 01390000
*             - Data areas       : None                             * 01400000
*             - Control blocks   : None                             * 01410000
*                                                                   * 01420000
```

```
 *                                                                    *
 *  Pseudocode:                                                       *
 *   DSN8DUCD:                                                        *
 *   - Issue sqlstate 38601 and a diagnostic message if no input date *
 *     was provided.                                                  *
 *   - Issue sqlstate 38601 and a diagnostic message if no input for- *
 *     mat was provided.                                              *
 *   - Issue sqlstate 38601 and a diagnostic message if no output     *
 *     format was provided.                                           *
 *   - Call deconDate to deconstruct the input date into year, month, *
 *     and day components according to the input format.              *
 *   - Call reconDate to create an output date from the  year, month, *
 *     and day components according to the output format.             *
 *   - If no errors, unset null indicators, and return SQLSTATE 00000 *
 *     else set null indicator and return null date out.              *
 *   End DSN8DUCD                                                     *
 *                                                                    *
 *   deconDate                                                        *
 *   - Parse day, month, and year (sequence unknown) components from  *
 *     the input date by breaking on delimiters (blank, /, ., and -). *
 *   - Use the input format to determine sequence of date components. *
 *     - if format invalid, issue SQLSTATE 38602 and a diag. message  *
 *   - Call checkDay to validate the day component                    *
 *     - if not valid day, issue SQLSTATE 38602 and a diag. message   *
 *   - Call checkMonth to validate the month component and convert it *
 *     (if required) from a calendar month name or roman numeral to   *
 *     a month number (1-12).                                         *
 *     - if not valid month, issue SQLSTATE 38602 and a diag. message *
 *   - Call checkYear to validate the year component.                 *
 *     - if not valid year, issue SQLSTATE 38602 and a diag. message  *
 *   End deconDate                                                    *
 *                                                                    *
 *   reconDate                                                        *
 *   - Use the output format to edit and sequence the date components *
 *     - call add0prefix to prepend leading 0's to the day and/or     *
 *       month component(s), as appropriate                           *
 *     - or call remove0prefix to drop leading 0's from the day and/  *
 *       or month component(s), as appropriate                        *
 *     - call nameMonth to convert the month number (1-12) to calen-  *
 *       dar name, if appropriate                                     *
 *     - call romanMonth to convert the month number (1-12) to roman  *
 *       numeral, if appropriate                                      *
 *     - call addCentury to convert a non-century year to century     *
 *       date, if appropriate                                         *
 *     - call removeCentury to convert a century year to non-century  *
 *       if appropriate                                               *
 *       - convert the month to a calendar name or roman numeral, if  *
 *         appropriate                                                *
 *       - convert the year to a non-century format, if appropriate   *
 *     - if output format is invalid, issue SQLSTATE 38603 and a      *
 *       diagnostic message                                           *
 *   - Call buildDate to create the output date from the edited, re-  *
 *     sequenced date components                                      *
 *   End reconDate                                                    *
 *                                                                    *
 *   buildDate                                                        *
 *   - Generate the date out by concatenating the date componentents  *
 *     (month, day, and year) with intervening delimiters (blank, .,  *
 *     /, or -) in the sequence directed by reconDate                 *
 *   End buildDate                                                    *
 *                                                                    *
 *   nameMonth                                                        *
 *   - convert a month in the standard form, 1-12, to the correspond- *
 *     ing caldendar month name.                                      *
 *   End nameMonth                                                    *
 *                                                                    *
 *   unnameMonth                                                      *
 *   - convert a calendar month name to the corresponding month no.   *
 *     in the standard form, 1-12.                                    *
 *   End unnameMonth                                                  *
 *                                                                    *
 *   romanMonth                                                       *
 *   - convert a month in the standard form, 1-12, to the correspond- *
 *     ing roman numeral, I-XII.                                      *
 *   End romanMonth                                                   *
 *                                                                    *
 *   unromanMonth                                                     *
 *   - convert a roman numeral (I-XII) to the corresponding month no. *
 *     in the standard form, 1-12.                                    *
 *   End unromanMonth                                                 *
 *                                                                    *
 *   checkYear                                                        *
```

```
*   - Verify that the year component of the input date is one of the *
*     following, in accordance with the input format:                *
*     - A valid century year (0000-9999)                             *
*     - A valid non-century year (00-99)                             *
*   - If not valid, set error flag and return null value for year    *
*   End checkYear                                                    *
*                                                                    *
*   checkMonth                                                       *
*   - Verify the month component of the input date in accordance     *
*     with the input format:                                         *
*     - if the month is a calendar name, call unnameMonth to convert *
*       it to a month number (1-12).                                 *
*     - if the month is a roman numeral, call unromanMonth to con-   *
*       vert it to a month number (1-12).                            *
*   - If not valid, set error flag and return null value for month   *
*   End checkMonth                                                   *
*                                                                    *
*   checkDay                                                         *
*   - Verify that the day component of the input date is one or two  *
*     numeric characters                                             *
*   - If not valid, set error flag and return null value for day     *
*   End checkDay                                                     *
*                                                                    *
*   add0prefix                                                       *
*   - prepend a day or month with a leading 0 if it is less than 10  *
*   End add0prefix                                                   *
*                                                                    *
*   remove0prefix                                                    *
*   - strip leading zero from a day or month if it is less than 10   *
*   End remove0prefix                                                *
*                                                                    *
*   addCentury                                                       *
*   - If the year component is non-century format, prepend it with   *
*     the current century.                                           *
*   End addCentury                                                   *
*                                                                    *
*   removeCentury                                                    *
*   - If the year component is century format, strip off the century *
*     portion.                                                       *
*   End removeCentury                                                *
*                                                                    *
*                                                                    *
*********************************************************************/

#pragma linkage(DSN8DUCD,fetchable)

/********************** C library definitions ***********************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

/***************************** Equates ******************************/
#define     NULLCHAR     '\0'         /* Null character            */

#define     MATCH         0           /* Comparison status: Equal   */
#define     NOT_OK        0           /* Run status indicator: Error*/
#define     OK            1           /* Run status indicator: Good */


/************************* Global constants ************************/
char        *char0 = "0";

char        *delimiters               /* Valid format delimiters    */
            = " .-/";

char        *monthNames[12]           /* Month names                */
            = { "January",  "February", "March",
                "April",    "May",      "June",
                "July",     "August",   "September",
                "October",  "November", "December" };

char        *monthNums[12]            /* Month numbers (as strings) */
            = { "1",        "2",        "3",
                "4",        "5",        "6",
                "7",        "8",        "9",
                "10",       "11",       "12" };

char        *romanNums[12]            /* Roman numerals             */
            = { "I",        "II",       "III",
                "IV",       "V",        "VI",
                "VII",      "VIII",     "IX",
```

```
                  "X",         "XI",         "XII" };


/*********************** DSN8DUCD functions ************************/
void DSN8DUCD                      /* main routine                 */
( char         *dateIn,            /* in: date to be converted     */
  char         *formatIn,          /* in: format of dateIn         */
  char         *formatOut,         /* in: format for dateOut       */
  char         *dateOut,           /* out: reformatted date        */
  short int    *nullDateIn,        /* in: indic var for dateIn     */
  short int    *nullFormatIn,      /* in: indic var for formatIn   */
  short int    *nullFormatOut,     /* in: indic var, formatOut     */
  short int    *nullDateOut,       /* out: indic var for dateOut   */
  char         *sqlstate,          /* out: SQLSTATE                */
  char         *fnName,            /* in: family name of function*/
  char         *specificName,      /* in: specific name of func    */
  char         *message            /* out: diagnostic message      */
);

int deconDate                      /* get yr, mo, dy from dateIn */
( char         *yr,                /* out: year component          */
  char         *mo,                /* out: month component         */
  char         *dy,                /* out: day component           */
  char         *message,           /* out: diagnostic message      */
  char         *sqlstate,          /* out: SQLSTATE                */
  char         *dateIn,            /* in: inputted date string     */
  char         *fmtIn              /* in: format of dateIn         */
);

int reconDate                      /* get dateOut from yr,mo,dy    */
( char         *dateOut,           /* out: reformatted date str    */
  char         *message,           /* out: diagnostic message      */
  char         *sqlstate,          /* out: SQLSTATE                */
  char         *yr,                /* in: year component           */
  char         *mo,                /* in: month component          */
  char         *dy,                /* in: day component            */
  char         *fmtOut             /* in: format for dateOut       */
);

void buildDate                     /* build date from parts        */
( char         *dtOut,             /* out: date                    */
  char         *d1,                /* in: year, month, or day      */
  char         *d2,                /* in: year, month, or day      */
  char         *d3,                /* in: year, month, or day      */
  char         *delim              /* in: delimiter                */
);

void add0prefix                    /* add leading zero to string */
( char         *str3               /* in/out: string to prefix     */
);

void remove0prefix                 /* strips leading zeroes        */
( char         *string             /* in/out: string to strip      */
);

int nameMonth                      /* converts month num to name */
( char         *monthIn            /* in/out: month to convert     */
);

int unnameMonth                    /* converts month name to num */
( char         *monthIn            /* in/out: month to convert     */
);

int romanMonth                     /* converts month# to roman#    */
( char         *monthIn            /* in/out: month to convert     */
);

int unromanMonth                   /* converts roman# to month#    */
( char         *monthIn            /* in/out: month to convert     */
);

int checkYear                      /* verify/standardize yearIn    */
( char         *yearOut,           /* out: 4-digit yr, validated */
  char         *yearIn,            /* in: 2- or 4-digit year       */
  char         *style              /* in: style of yearIn          */
);

int checkMonth                     /* verify/standardize monthIn */
( char         *monthOut,          /* out: month#, validated       */
  char         *monthIn,           /* in: month name, #, roman#    */
  char         *style              /* in: style of monthIn         */
);
```

```

int checkDay                         /* verify/standardize dayIn   */
( char        *dayOut,               /* out: day, validated        */
  char        *dayIn                 /* in: day number             */
);

void addCentury                      /* adds century to yearIn     */
( char        *yearIn                /* in/out: year               */
);

void removeCentury                   /* strip century from yearIn  */
( char        *yearIn                /* in/out: year               */
);

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
void DSN8DUCD                        /* main routine               */
( char        *dateIn,               /* in: date to be converted   */
  char        *formatIn,             /* in: format of dateIn       */
  char        *formatOut,            /* in: format for dateOut     */
  char        *dateOut,              /* out: reformatted date      */
  short int   *nullDateIn,           /* in: indic var for dateIn   */
  short int   *nullFormatIn,         /* in: indic var for formatIn */
  short int   *nullFormatOut,        /* in: indic var, formatOut   */
  short int   *nullDateOut,          /* out: indic var for dateOut */
  char        *sqlstate,             /* out: SQLSTATE              */
  char        *fnName,               /* in: family name of function*/
  char        *specificName,         /* in: specific name of func  */
  char        *message               /* out: diagnostic message    */
)
/*******************************************************************
*                                                                  *
* Assumptions:                                                     *
* - *dateIn        points to a char[18], null-terminated string    *
* - *formatIn,     points to a char[14], null-terminated string    *
* - *formatOut     points to a char[14], null-terminated string    *
* - *dateOut       points to a char[18], null-terminated string    *
* - *nullDateIn    points to a short integer                       *
* - *nullFormatIn  points to a short integer                       *
* - *nullFormatOut points to a short integer                       *
* - *nullDateOut   points to a short integer                       *
* - *sqlstate      points to a char[06], null-terminated string    *
* - *fnName        points to a char[138], null-terminated string   *
* - *specificName  points to a char[129], null-terminated string   *
* - *message       points to a char[70], null-terminated string    *
*******************************************************************/
{
  /************************ Local variables ************************/
  short int   i;                     /* loop control vars          */
  char        year[5];               /* gets year from dateIn      */
  char        month[10];             /* gets month from dateIn     */
  char        day[3];                /* gets day from dateIn       */

  short int   status = OK;           /* DSN8DUCD run status        */

  /*****************************************************************
  * Verify that an input date, its current format, and its new format*
  * have been passed in.                                           *
  *****************************************************************/
  if( *nullDateIn || ( strlen( dateIn ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCD Error: No input date entered" );
      strcpy( sqlstate, "38601" );
    }
  else if( *nullFormatIn || ( strlen( formatIn ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCD Error: No input format entered" );
      strcpy( sqlstate, "38601" );
    }
  else if( *nullFormatOut || ( strlen( formatOut ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCD Error: No output format entered" );
      strcpy( sqlstate, "38601" );
    }

```

```
  /********************************************************************
  * Use formatIn to deconstruct date in into year, month, and day    *
  ********************************************************************/
  if( status == OK )
    status = deconDate( year, month, day, message, sqlstate,
                        dateIn, formatIn );

  /********************************************************************
  * Use formatOut to reconstruct date from year, month, and day      *
  ********************************************************************/
  if( status == OK )
    status = reconDate( dateOut, message, sqlstate,
                        year, month, day, formatOut );

  /********************************************************************
  * If conversion was successful, clear the message buffer and sql-  *
  * state, and unset the SQL null indicator for dateOut.             *
  ********************************************************************/
  if( status == OK )
    {
      *nullDateOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /********************************************************************
  * If errors occurred, clear the dateOut buffer and set the SQL null*
  * indicator.  A diagnostic message and the SQLSTATE have been set  *
  * where the error was detected.                                    *
  ********************************************************************/
  else
    {
      dateOut[0] = NULLCHAR;
      *nullDateOut = -1;
    }

  return;
} /* end of DSN8DUCD */


/*********************************************************************/
/****************************** Functions ****************************/
/*********************************************************************/
int deconDate                         /* get yr, mo, dy from dateIn */
( char        *yr,                    /* out: year component        */
  char        *mo,                    /* out: month component       */
  char        *dy,                    /* out: day component         */
  char        *message,               /* out: diagnostic message    */
  char        *sqlstate,              /* out: SQLSTATE              */
  char        *dateIn,                /* in: inputted date string   */
  char        *fmtIn                  /* in: format of dateIn       */
)
/**********************************************************************
* Deconstructs *dateIn into *yr, *mo, and *dy according to *fmtIn.   *
* Returns 1 if deconstruction succeeds, otherwise places diagnostic  *
* text in *message and returns 0.                                    *
**********************************************************************/
{
  /************************ Local variables ************************/

  short int   func_status = OK;       /* function status indicator  */
  short int   yrStatus = OK;          /* indicates if year    is OK */
  short int   moStatus = OK;          /*     "     "  month   "  "  */
  short int   dyStatus = OK;          /*     "     "  day     "  "  */
  short int   ftStatus = OK;          /*     "     "  format  "  "  */

  char        workDateIn[18];         /* work copy of dateIn        */
  char        *token;                 /* Value from token parser    */

  char        tok1[17];               /* Gets 1st date component    */
  char        tok2[17];               /*  "   2nd  "       "        */
  char        tok3[17];               /*  "   3rd  "       "        */

  /********************************************************************
  * Parse day, month, and year (order unknown) from dateIn           *
  ********************************************************************/
  strcpy( workDateIn,dateIn );
  token = strtok( workDateIn," .-/" );
  strcpy( tok1,token );
  token = strtok( NULL," .-/" );
  strcpy( tok2,token );
  token = strtok( NULL," .-/" );
  strcpy( tok3,token );
```

```

/******************************************************************
* Use fmtIn to check and set year, month, and day from date tokens *
******************************************************************/
if( ( strcmp( fmtIn,"D MONTH YY" ) == MATCH )
 || ( strcmp( fmtIn,"DD MONTH YY" ) == MATCH ) )
  {
    dyStatus = checkDay( dy,tok1 );
    moStatus = checkMonth( mo,tok2,"MONTH" );
    yrStatus = checkYear( yr,tok3,"YY" );
  }
else if( ( strcmp( fmtIn,"D MONTH YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"DD MONTH YYYY" ) == MATCH ) )
  {
    dyStatus = checkDay( dy,tok1 );
    moStatus = checkMonth( mo,tok2,"MONTH" );
    yrStatus = checkYear( yr,tok3,"YYYY" );
  }
else if( ( strcmp( fmtIn,"D.M.YY" ) == MATCH )
      || ( strcmp( fmtIn,"DD.MM.YY" ) == MATCH )
      || ( strcmp( fmtIn,"D-M-YY" ) == MATCH )
      || ( strcmp( fmtIn,"DD-MM-YY" ) == MATCH )
      || ( strcmp( fmtIn,"D/M/YY" ) == MATCH )
      || ( strcmp( fmtIn,"DD/MM/YY" ) == MATCH ) )
  {
    dyStatus = checkDay( dy,tok1 );
    moStatus = checkMonth( mo,tok2,"M/MM" );
    yrStatus = checkYear( yr,tok3,"YY" );
  }
else if( ( strcmp( fmtIn,"D.M.YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"DD.MM.YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"D-M-YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"DD-MM-YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"D/M/YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"DD/MM/YYYY" ) == MATCH ) )
  {
    dyStatus = checkDay( dy,tok1 );
    moStatus = checkMonth( mo,tok2,"M/MM" );
    yrStatus = checkYear( yr,tok3,"YYYY" );
  }
else if( ( strcmp( fmtIn,"M/D/YY" ) == MATCH )
      || ( strcmp( fmtIn,"MM/DD/YY" ) == MATCH ) )
  {
    moStatus = checkMonth( mo,tok1,"M/MM" );
    dyStatus = checkDay( dy,tok2 );
    yrStatus = checkYear( yr,tok3,"YY" );
  }
else if( ( strcmp( fmtIn,"M/D/YYYY" ) == MATCH )
      || ( strcmp( fmtIn,"MM/DD/YYYY" ) == MATCH ) )
  {
    moStatus = checkMonth( mo,tok1,"M/MM" );
    dyStatus = checkDay( dy,tok2 );
    yrStatus = checkYear( yr,tok3,"YYYY" );
  }
else if( ( strcmp( fmtIn,"YY/M/D" ) == MATCH )
      || ( strcmp( fmtIn,"YY/MM/DD" ) == MATCH )
      || ( strcmp( fmtIn,"YY.M.D" ) == MATCH )
      || ( strcmp( fmtIn,"YY.MM.DD" ) == MATCH ) )
  {
    yrStatus = checkYear( yr,tok1,"YY" );
    moStatus = checkMonth( mo,tok2,"M/MM" );
    dyStatus = checkDay( dy,tok3 );
  }
else if( ( strcmp( fmtIn,"YYYY/M/D" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY/MM/DD" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY.M.D" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY.MM.DD" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY-M-D" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY-MM-DD" ) == MATCH ) )
  {
    yrStatus = checkYear( yr,tok1,"YYYY" );
    moStatus = checkMonth( mo,tok2,"M/MM" );
    dyStatus = checkDay( dy,tok3 );
  }
else if( ( strcmp( fmtIn,"YYYY-D-XX" ) == MATCH )
      || ( strcmp( fmtIn,"YYYY-DD-XX" ) == MATCH ) )
  {
    yrStatus = checkYear( yr,tok1,"YYYY" );
    dyStatus = checkDay( dy,tok2 );
    moStatus = checkMonth( mo,tok3,"XX" );
  }
else if( ( strcmp( fmtIn,"YYYY-XX-D" ) == MATCH )
```

```
        || ( strcmp( fmtIn,"YYYY-XX-DD" ) == MATCH ) )
    {
      yrStatus = checkYear( yr,tok1,"YYYY" );
      moStatus = checkMonth( mo,tok2,"XX" );
      dyStatus = checkDay( dy,tok3 );
    }
  else /* date-in format is invalid or unknown */
    ftStatus = NOT_OK;

  /*****************************************************************
  * set up error handling                                          *
  *****************************************************************/
  func_status = NOT_OK;
  strcpy( message,"DSN8DUCD Error: " );
  strcpy( sqlstate, "38602" );

  /*****************************************************************
  * if error detected, issue diagnostic message and return NOT_OK  *
  *****************************************************************/
  if( ftStatus != OK )
    strcpy( message,
            "Unknown input format specified" );
  else if( yrStatus != OK )
    strcpy( message,
            "Value for year "
            "is incorrect or does not "
            "conform to input format" );
  else if( moStatus != OK )
    strcpy( message,
            "Value for month "
            "is incorrect or does not "
            "conform to input format " );
  else if( dyStatus != OK )
    strcpy( message,
            "Value for day "
            "is incorrect or does not "
            "conform to input format " );

  /*****************************************************************
  * if no error detected, clear message and sqlstate and return OK *
  *****************************************************************/
  else
    {
      *message = NULLCHAR;
      func_status = OK;
      strcpy( sqlstate, "00000" );
    }

  return( func_status );
}  /* end deconDate */

int reconDate                        /* get dateOut from yr,mo,dy  */
( char        *dateOut,              /* out: reformatted date str  */
  char        *message,              /* out: diagnostic message    */
  char        *sqlstate,             /* out: SQLSTATE              */
  char        *yr,                   /* in: year component         */
  char        *mo,                   /* in: month component        */
  char        *dy,                   /* in: day component          */
  char        *fmtOut                /* in: format for dateOut     */
)
/*******************************************************************
* Reconstructs *yr, *mo, and *dy into *dateOut according to *fmtOut. *
* Returns 1 if reconstruction succeeds, otherwise places diagnostic  *
* text in *message and returns 0.                                    *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   func_status = OK;      /* function status indicator  */

  /*****************************************************************
  * Use fmtOut to reformat date from year, month, and day tokens   *
  *****************************************************************/
  if( strcmp( fmtOut,"D MONTH YY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      nameMonth( mo );               /* convert month no. to name  */
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, " " );
    }
  else if( strcmp( fmtOut,"DD MONTH YY" ) == MATCH )
    {
```

```
      add0prefix( dy );              /* add leading 0 if day < 10  */
      nameMonth( mo );               /* convert month no. to name  */
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, " " );
    }
  else if( strcmp( fmtOut,"D MONTH YYYY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      nameMonth( mo );               /* convert month no. to name  */
      addCentury( yr );              /* ensure year has century    */
      buildDate( dateOut, dy, mo, yr, " " );
    }
  else if( strcmp( fmtOut,"DD MONTH YYYY" ) == MATCH )
    {
      add0prefix( dy );              /* add leading 0 if day < 10  */
      nameMonth( mo );               /* convert month no. to name  */
      addCentury( yr );              /* ensure year has century    */
      buildDate( dateOut, dy, mo, yr, " " );
    }
  else if( strcmp( fmtOut,"D.M.YY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      remove0prefix( mo );           /* strip leading 0 if mon < 10*/
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "." );
    }
  else if( strcmp( fmtOut,"DD.MM.YY" ) == MATCH )
    {
      add0prefix( dy );              /* add leading 0 if day < 10  */
      add0prefix( mo );              /* add leading 0 if mon < 10  */
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "." );
    }
  else if( strcmp( fmtOut,"D-M-YY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      remove0prefix( mo );           /* strip leading 0 if mon < 10*/
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "-" );
    }
  else if( strcmp( fmtOut,"DD-MM-YY" ) == MATCH )
    {
      add0prefix( dy );              /* add leading 0 if day < 10  */
      add0prefix( mo );              /* add leading 0 if mon < 10  */
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "-" );
    }
  else if( strcmp( fmtOut,"D/M/YY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      remove0prefix( mo );           /* strip leading 0 if mon < 10*/
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "/" );
    }
  else if( strcmp( fmtOut,"DD/MM/YY" ) == MATCH )
    {
      add0prefix( dy );              /* add leading 0 if day < 10  */
      add0prefix( mo );              /* add leading 0 if mon < 10  */
      removeCentury( yr );           /* strip century from year    */
      buildDate( dateOut, dy, mo, yr, "/" );
    }
  else if( strcmp( fmtOut,"D.M.YYYY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      remove0prefix( mo );           /* strip leading 0 if mon < 10*/
      addCentury( yr );              /* ensure year has century    */
      buildDate( dateOut, dy, mo, yr, "." );
    }
  else if( strcmp( fmtOut,"DD.MM.YYYY" ) == MATCH )
    {
      add0prefix( dy );              /* add leading 0 if day < 10  */
      add0prefix( mo );              /* add leading 0 if mon < 10  */
      addCentury( yr );              /* ensure year has century    */
      buildDate( dateOut, dy, mo, yr, "." );
    }
  else if( strcmp( fmtOut,"D-M-YYYY" ) == MATCH )
    {
      remove0prefix( dy );           /* strip leading 0 if day < 10*/
      remove0prefix( mo );           /* strip leading 0 if mon < 10*/
      addCentury( yr );              /* ensure year has century    */
      buildDate( dateOut, dy, mo, yr, "-" );
    }
```

```
    else if( strcmp( fmtOut,"DD-MM-YYYY" ) == MATCH )
      {
        add0prefix( dy );               /* add leading 0 if day < 10  */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        addCentury( yr );               /* ensure year has century    */
        buildDate( dateOut, dy, mo, yr, "-" );
      }
    else if( strcmp( fmtOut,"D/M/YYYY" ) == MATCH )
      {
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        addCentury( yr );               /* ensure year has century    */
        buildDate( dateOut, dy, mo, yr, "/" );
      }
    else if( strcmp( fmtOut,"DD/MM/YYYY" ) == MATCH )
      {
        add0prefix( dy );               /* add leading 0 if day < 10  */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        addCentury( yr );               /* ensure year has century    */
        buildDate( dateOut, dy, mo, yr, "/" );
      }
    else if( strcmp( fmtOut,"M/D/YY" ) == MATCH )
      {
        remove0prefix( mo );            /* strip leading 0 if day < 10*/
        remove0prefix( dy );            /* strip leading 0 if mon < 10*/
        removeCentury( yr );            /* strip century from year    */
        buildDate( dateOut, mo, dy, yr, "/" );
      }
    else if( strcmp( fmtOut,"MM/DD/YY" ) == MATCH )
      {
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        removeCentury( yr );            /* strip century from year    */
        buildDate( dateOut, mo, dy, yr, "/" );
      }
    else if( strcmp( fmtOut,"M/D/YYYY" ) == MATCH )
      {
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        addCentury( yr );               /* ensure year has century    */
        buildDate( dateOut, mo, dy, yr, "/" );
      }
    else if( strcmp( fmtOut,"MM/DD/YYYY" ) == MATCH )
      {
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        addCentury( yr );               /* ensure year has century    */
        buildDate( dateOut, mo, dy, yr, "/" );
      }
    else if( strcmp( fmtOut,"YY/M/D" ) == MATCH )
      {
        removeCentury( yr );            /* strip century from year    */
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        buildDate( dateOut, yr, mo, dy, "/" );
      }
    else if( strcmp( fmtOut,"YY/MM/DD" ) == MATCH )
      {
        removeCentury( yr );            /* strip century from year    */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "/" );
      }
    else if( strcmp( fmtOut,"YY.M.D" ) == MATCH )
      {
        removeCentury( yr );            /* strip century from year    */
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        buildDate( dateOut, yr, mo, dy, "." );
      }
    else if( strcmp( fmtOut,"YY.MM.DD" ) == MATCH )
      {
        removeCentury( yr );            /* strip century from year    */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "." );
      }
    else if( strcmp( fmtOut,"YYYY/M/D" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
```

```
        buildDate( dateOut, yr, mo, dy, "/" );
      }
    else if( strcmp( fmtOut,"YYYY/MM/DD" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "/" );
      }
    else if( strcmp( fmtOut,"YYYY.M.D" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        buildDate( dateOut, yr, mo, dy, "." );
      }
    else if( strcmp( fmtOut,"YYYY.MM.DD" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "." );
      }
    else if( strcmp( fmtOut,"YYYY-M-D" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        remove0prefix( mo );            /* strip leading 0 if mon < 10*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        buildDate( dateOut, yr, mo, dy, "-" );
      }
    else if( strcmp( fmtOut,"YYYY-MM-DD" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        add0prefix( mo );               /* add leading 0 if mon < 10  */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "-" );
      }
    else if( strcmp( fmtOut,"YYYY-D-XX" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        romanMonth( mo );               /* convert month# to roman no.*/
        buildDate( dateOut, yr, dy, mo, "-" );
      }
    else if( strcmp( fmtOut,"YYYY-DD-XX" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        add0prefix( dy );               /* add leading 0 if day < 10  */
        romanMonth( mo );               /* convert month# to roman no.*/
        buildDate( dateOut, yr, dy, mo, "-" );
      }
    else if( strcmp( fmtOut,"YYYY-XX-D" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        romanMonth( mo );               /* convert month# to roman no.*/
        remove0prefix( dy );            /* strip leading 0 if day < 10*/
        buildDate( dateOut, yr, mo, dy, "-" );
      }
    else if( strcmp( fmtOut,"YYYY-XX-DD" ) == MATCH )
      {
        addCentury( yr );               /* ensure year has century    */
        romanMonth( mo );               /* convert month# to roman no.*/
        add0prefix( dy );               /* add leading 0 if day < 10  */
        buildDate( dateOut, yr, mo, dy, "-" );
      }
    else /* date-in format is invalid or unknown */
      func_status = NOT_OK;

    if( func_status != OK )
      {
        strcpy( sqlstate, "38603" );
        strcpy( message,
                "Unknown output format specified" );
      }
    else
      {
        *message = NULLCHAR;
        strcpy( sqlstate, "00000" );
      }

    return( func_status );
  } /* end reconDate */
```

```

void buildDate                        /* build date from parts      */
( char        *dtOut,                 /* out: date                  */
  char        *d1,                    /* in: year, month, or day    */
  char        *d2,                    /* in: year, month, or day    */
  char        *d3,                    /* in: year, month, or day    */
  char        *delim                  /* in: delimiter              */
)
/*********************************************************************
* Forms a date by concatenating d1, delim, d2, delim, and d3.        *
*********************************************************************/
{
  strcpy( dtOut, d1 );
  strcat( dtOut, delim );
  strcat( dtOut, d2 );
  strcat( dtOut, delim );
  strcat( dtOut, d3 );
} /* end buildDate */


int nameMonth                         /* converts month num to name */
( char        *monthIn                /* in/out: month to convert   */
)
/*********************************************************************
* Converts *monthIn from a number string to a name.  Returns 1 if    *
* conversion succeeds, otherwise returns 0.                          *
*********************************************************************/
{
  /************************ Local variables *************************/

  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * Strip leading zero (if any) from monthIn                         *
  *******************************************************************/
  remove0prefix( monthIn );

  /*******************************************************************
  * Look up *monthIn in the month number strings array               *
  *******************************************************************/
  for( i=0; i<12 && strcmp( monthIn,monthNums[i] ) != MATCH; i++ );

  /*******************************************************************
  * If found assign month name else set function error indicator     *
  *******************************************************************/
  if( i < 12 )
    strcpy( monthIn,monthNames[i] );
  else
   func_status = NOT_OK;

  return( func_status );

} /* end nameMonth */


int unnameMonth                       /* converts month name to num */
( char        *monthIn                /* in/out: month to convert   */
)
/*********************************************************************
* Converts *monthIn from a name to a number string.  Returns 1 if    *
* conversion succeeds, otherwise returns 0.                          *
*********************************************************************/
{
  /************************ Local variables *************************/

  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * Make 1st char of month name upper case and the rest lower case   *
  *******************************************************************/
  monthIn[0] = toupper(monthIn[0]);
  for( i=1;i<strlen(monthIn);i++ )
    monthIn[i] = tolower(monthIn[i]);

  /*******************************************************************
  * Look up *monthIn in the month names array                        *
  *******************************************************************/
  for( i=0; i<12 && strcmp( monthIn,monthNames[i] ) != MATCH; i++ );

```

```
  /****************************************************************
  * If found assign month no. str  else set function error indicator *
  *****************************************************************/
  if( i < 12 )
    strcpy( monthIn,monthNums[i] );
  else
   func_status = NOT_OK;

  return( func_status );

} /* end unnameMonth */


int romanMonth                        /* converts month# to roman#  */
( char       *monthIn                 /* in/out: month to convert   */
)
/*******************************************************************
* Converts *monthIn from a number string to a roman numeral. Returns *
* 1 if conversion succeeds, otherwise returns 0.                   *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /****************************************************************
  * Strip leading zero (if any) from monthIn                        *
  *****************************************************************/
  remove0prefix( monthIn );

  /****************************************************************
  * Look up *monthIn in the month number strings array              *
  *****************************************************************/
  for( i=0; i<12 && strcmp( monthIn,monthNums[i] ) != MATCH; i++ );

  /****************************************************************
  * If found assign roman numeral else set function error indicator  *
  *****************************************************************/
  if( i < 12 )
    strcpy( monthIn,romanNums[i] );
  else
   func_status = NOT_OK;

  return( func_status );

} /* end romanMonth */


int unromanMonth                      /* converts roman# to month#  */
( char       *monthIn                 /* in/out: month to convert   */
)
/*******************************************************************
* Converts *monthIn from a roman numeral to a number string. Returns *
* 1 if conversion succeeds, otherwise returns 0.                   *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /****************************************************************
  * Convert all chars of *monthIn to upper case                     *
  *****************************************************************/
  for( i=0; i<strlen(monthIn); i++ )
    monthIn[i] = toupper(monthIn[i]);

  /****************************************************************
  * Look up *monthIn in the roman numerals array                    *
  *****************************************************************/
  for( i=0; i<12 && strcmp( monthIn,romanNums[i] ) != MATCH; i++ );

  /****************************************************************
  * If found assign month no. str  else set function error indicator *
  *****************************************************************/
  if( i < 12 )
    strcpy( monthIn,monthNums[i] );
  else
   func_status = NOT_OK;

  return( func_status );
```

```

} /* end unromanMonth */


int checkYear                        /* verify/standardize yearIn  */
( char        *yearOut,              /* out: 4-digit yr, validated */
  char        *yearIn,               /* in: 2- or 4-digit year     */
  char        *style                 /* in: style of yearIn        */
)
/*******************************************************************
* Verifies that *yearIn is either of the following:                *
* - 2 numeric characters if *style is YY; or                       *
* - 4 numeric characters if *style is YYYY.                        *
* If criteria satisfied, copies *yearIn to *yearOut and returns 1.   *
* If criteria not satisfied, sets *yearOut to null and returns 0.    *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   i;                     /* loop control               */
  short int   yearIn_len             /* length of *yearIn          */
              = strlen( yearIn );
  short int   func_status = OK;      /* function status indicator  */

  /*****************************************************************
  * Verify that all bytes of *yearIn are numeric characters        *
  *****************************************************************/
  for( i=0; (i<yearIn_len) && (isdigit(yearIn[i])); i++ );
  if( i < yearIn_len )
    func_status = NOT_OK;
  /*****************************************************************
  * If input format is YY, verify that *yearIn has 2 bytes         *
  *****************************************************************/
  else if( (strcmp( style,"YY" ) == MATCH) && (yearIn_len != 2) )
    func_status = NOT_OK;
  /*****************************************************************
  * If input format is YYYY, verify that *yearIn has 4 bytes       *
  *****************************************************************/
  else if( (strcmp( style,"YYYY" ) == MATCH) && (yearIn_len != 4) )
    func_status = NOT_OK;

  /*****************************************************************
  * If all checks satisfied, copy *yearIn to *yearOut and return 1 *
  *****************************************************************/
  if( func_status == OK )
    strcpy( yearOut, yearIn );
  /*****************************************************************
  * If a check failed, sets *yearOut to null and return 0          *
  *****************************************************************/
  else
    *yearOut = NULLCHAR;

  return( func_status );
} /* end checkYear */


int checkMonth                       /* verify/standardize monthIn */
( char        *monthOut,             /* out: month#, validated     */
  char        *monthIn,              /* in: month name, #, roman#  */
  char        *style                 /* in: style of monthIn       */
)
/*******************************************************************
* Verifies that *monthIn is one of the following:                  *
* - A valid month name, January - December, if *style is MONTH; or *
* - A valid roman numeral, I - XII, if *style is XX; or            *
* - 1 or 2 numeric characters between 1 and 12 if *style is M or MM. *
* If criteria satisfied, copies *monthIn to *monthOut and returns 1. *
* - if *monthIn is a month name or a roman numeral, it will have   *
*   been standardized to the form 1-12.                            *
* If criteria not satisfied, sets *monthOut to null and returns 0.   *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   i;                     /* loop control               */
  short int   func_status = OK;      /* function status indicator  */

  /*****************************************************************
  * If *style is MONTH, verify that *monthIn is a valid month name *
  *****************************************************************/
  if( strcmp( style,"MONTH" ) == MATCH )
    func_status = unnameMonth( monthIn );
```

```
  /*****************************************************************
  * If *style is XX, verify that *monthIn is a roman numeral, I - XII*
  *****************************************************************/
  else if( strcmp( style,"XX" ) == MATCH )
    func_status = unromanMonth( monthIn );
  /*****************************************************************
  * Otherwise, verify that *monthIn is valid month number, 1 - 12    *
  *****************************************************************/
  else
    {
      remove0prefix( monthIn );       /* strip any leading zero     */
      for( i=0; i<12 && strcmp( monthIn,monthNums[i] ) != MATCH; i++ );
      if( i >= 12 )
        func_status = NOT_OK;
    }
  /*****************************************************************
  * If all checks satisfied, copy *monthIn to *monthOut and return 1 *
  *****************************************************************/
  if( func_status == OK )
    strcpy( monthOut, monthIn );
  /*****************************************************************
  * If a check failed, set *monthOut to null and return 0            *
  *****************************************************************/
  else
    *monthOut = NULLCHAR;

  return( func_status );
} /* end checkMonth */


int checkDay                          /* verify/standardize dayIn   */
( char        *dayOut,                /* out: day, validated        */
  char        *dayIn                  /* in: day number             */
)
/*******************************************************************
* Verifies that *dayIn is either 1 or 2 numeric characters.        *
* If criteria satisfied, copies *dayIn to *dayOut and returns 1.    *
* If criteria not satisfied, set *dayOut to null and returns 0.     *
*******************************************************************/
{
  /************************ Local variables ************************/

  short int   i;                      /* loop control               */
  short int   dayIn_len               /* length of *dayIn           */
              = strlen( dayIn );
  short int   func_status = OK;       /* function status indicator  */

  /*****************************************************************
  * Verify that *dayIn is 1 or 2 numeric characters                  *
  *****************************************************************/
  for( i=0; ( i<dayIn_len ) && ( isdigit(dayIn[i]) ); i++ );
  if( i < dayIn_len || dayIn_len < 1 || dayIn_len > 2 )
    func_status = NOT_OK;
  /*****************************************************************
  * If all checks satisfied, copy *dayIn to *dayOut and return 1     *
  *****************************************************************/
  if( func_status == OK )
    strcpy( dayOut, dayIn );
  /*****************************************************************
  * If a check failed, set *dayOut to null and return 0              *
  *****************************************************************/
  else
    *dayOut = NULLCHAR;

  return( func_status );
} /* end checkDay */


void add0prefix
( char        *str3                   /* in/out: string to prefix   */
)
/*******************************************************************
* Prefixes *str3 with a leading 0 if it is only 1 byte long.        *
*******************************************************************/
{
  /************************ Local variables ************************/

  if( strlen( str3 ) == 1 )
    {
      str3[1] = str3[0];              /* Right-shift *str3 1 byte   */
      str3[0] = *char0;               /* Prefix it with "0"         */
      str3[2] = NULLCHAR;             /* And terminate it           */
```

```
    }

} /* end add0prefix */


void remove0prefix                    /* strips leading zeroes      */
( char        *string                 /* in/out: string to strip    */
)
/*******************************************************************
* Strips the leading zero from *string, if it has one.             *
*******************************************************************/
{
  if( strncmp( string,"0",1 ) == MATCH )
    {
      string[0] = string[1];          /* Left-shift *string         */
      string[1] = NULLCHAR;           /* And terminate it           */
    }

} /* end remove0prefix */


void addCentury                       /* adds century to yearIn     */
( char        *yearIn                 /* in/out: year               */
)
/*******************************************************************
* Prefixes *yearIn with the current century (according to the system *
* date) if *yearIn is 2 bytes long.                                *
*******************************************************************/
{
  /************************ Local variables ************************/

  time_t      t;                      /* receives calendar time     */
  struct  tm  *timeptr;               /* recieves local time        */
  char        centyear[4];            /* receives current century   */


  /*****************************************************************
  * If *yearIn is 2 bytes long, prefix it with the current century   *
  *****************************************************************/
  if( strlen( yearIn ) == 2 )
    {
      t = time(NULL);                 /* Get calendar time from sys */
      timeptr = localtime(&t);        /* Convert to local time      */
      strftime( centyear,             /* Format current century year*/
                sizeof(centyear)-1,   /* ..sized for receiving field*/
                "%Y",                 /* ..as century year          */
                timeptr );            /* ..from current local time  */

                                      /* Prefix *yearIn with century*/
      yearIn[3] = yearIn[1];          /* ..Right-shift *yearIn      */
      yearIn[2] = yearIn[0];          /* ..by 2 bytes               */
      yearIn[1] = centyear[1];        /* ..Place the century portion*/
      yearIn[0] = centyear[0];        /* ..in bytes 1-2             */
      yearIn[4] = NULLCHAR;           /* ..Terminate the string     */
    }

} /* end addCentury */


void removeCentury                    /* strip century from yearIn  */
( char        *yearIn                 /* in/out: year               */
)
/*******************************************************************
* Strips the century portion from *yearIn if it consists of 4 bytes. *
*******************************************************************/
{
  /*****************************************************************
  * If *yearIn is 4 bytes long, strip off the century portion       *
  *****************************************************************/
  if( strlen( yearIn ) == 4 )
    {
      yearIn[0] = yearIn[2];          /* Shift non-century portion  */
      yearIn[1] = yearIn[3];          /* of *yearIn to 1st 2 bytes  */
      yearIn[2] = NULLCHAR;           /* and terminate string       */
    }
} /* end removeCentury */
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8DUCT

Converts a given time from one to another of these 8 formats.

```
/******************************************************************** 00010000
* Module name = DSN8DUCT (DB2 sample program)                       * 00020000
*                                                                   * 00030000
* DESCRIPTIVE NAME = General time reformatter (UDF)                 * 00040000
*                                                                   * 00050000
*  LICENSED MATERIALS - PROPERTY OF IBM                             * 00060000
*  5675-DB2                                                         * 00109990
*  (C) COPYRIGHT 2000 IBM CORP.  ALL RIGHTS RESERVED.               * 00149980
*                                                                   * 00190000
*  STATUS = VERSION 7                                               * 00200000
*                                                                   * 00210000
* Function: Converts a given time from one to another of these 8    * 00220000
*           formats:                                                * 00230000
*                                                                   * 00240000
*             H:MM AM/PM   HH:MM AM/PM   HH:MM:SS AM/PM   HH:MM:SS   * 00250000
*             H.MM         HH.MM         H.MM.SS          HH.MM.SS   * 00260000
*                                                                   * 00270000
*           where:                                                  * 00280000
*                                                                   * 00290000
*             H: Suppress leading zero if the hour is less than 10   * 00300000
*            HH: Retain leading zero if the hour is less than 10     * 00310000
*             M: Suppress leading zero if the minute is less than 10 * 00320000
*            MM: Retain leading zero if the minute is less than 10   * 00330000
*         AM/PM: Return time in 12-hour clock format, else 24-hour   * 00340000
*                                                                   * 00350000
*           Example invocation:                                     * 00360000
*            EXEC SQL SET :then = ALTTIME( "01:34:59 PM",           * 00370000
*                                          "HH:MM:SS AM/PM",        * 00380000
*                                          "H.MM" );                * 00390000
*            ==> then = "13.34"                                     * 00400000
*                                                                   * 00410000
* Notes:                                                            * 00420000
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       * 00430000
*                                                                   * 00440000
*   Restrictions:                                                   * 00450000
*                                                                   * 00460000
* Module type: C program                                            * 00470000
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   * 00480000
* Module size: See linkedit output                                  * 00490000
*  Attributes: Re-entrant and re-usable                             * 00500000
*                                                                   * 00510000
* Entry Point: DSN8DUCT                                             * 00520000
*     Purpose: See Function                                         * 00530000
*     Linkage: DB2SQL                                               * 00540000
*              Invoked via SQL UDF call                             * 00550000
*                                                                   * 00560000
*       Input: Parameters explicitly passed to this function:       * 00570000
*              - *timeIn      : pointer to a char[12], null-termi-   * 00580000
*                               nated string having a time in the   * 00590000
*                               format indicated by *formatIn.      * 00600000
*              - *formatIn    : pointer to a char[15], null-termi-   * 00610000
*                               nated string having the format of   * 00620000
*                               time found in *timeIn (see "Func-   * 00630000
*                               tion", above, for valid formats).   * 00640000
*              - *formatOut   : pointer to a char[15], null-termi-   * 00650000
*                               nated string having the format to   * 00660000
*                               which the time found in *timeIn is   * 00670000
*                               to be converted.  See "Function",   * 00680000
*                               above, for valid formats.           * 00690000
*              - *niTimeIn    : pointer to a short integer having   * 00700000
*                               the null indicator variable for     * 00710000
*                               *timeIn.                            * 00720000
*              - *niFormatIn  : pointer to a short integer having   * 00730000
*                               the null indicator variable for     * 00740000
*                               *formatIn.                          * 00750000
*              - *niFormatOut : pointer to a short integer having   * 00760000
*                               the null indicator variable for     * 00770000
*                               *formatOut.                         * 00780000
*              - *fnName      : pointer to a char[138], null-termi-  * 00790000
*                               nated string having the UDF family   * 00800000
*                               name of this function.              * 00810000
*              - *specificName: pointer to a char[129], null-termi-  * 00820000
*                               nated string having the UDF specific * 00830000
```

```
 *                              name of this function.              *
 *                                                                  *
 *                                                                  *
 *      Output: Parameters explicitly passed by this function:      *
 *              - *timeOut     : pointer to a char[15], null-termi- *
 *                               nated string to receive the refor- *
 *                               matted time.                       *
 *              - *niTimeOut   : pointer to a short integer to re-  *
 *                               ceive the null indicator variable  *
 *                               for *timeOut.                      *
 *              - *sqlstate    : pointer to a char[06], null-termi- *
 *                               nated string to receive the SQLSTATE.*
 *              - *message     : pointer to a char[70], null-termi- *
 *                               nated string to receive a diagnostic *
 *                               message if one is generated by this *
 *                               function.                          *
 *                                                                  *
 * Normal Exit: Return Code: SQLSTATE = 00000                       *
 *              - Message: none                                     *
 *                                                                  *
 *  Error Exit: Return Code: SQLSTATE = 38601                       *
 *              - Message: DSN8DUCT Error: No input time entered    *
 *              - Message: DSN8DUCT Error: No input format entered  *
 *              - Message: DSN8DUCT Error: No output format entered *
 *                                                                  *
 *              Return Code: SQLSTATE = 38602                       *
 *              - Message: DSN8DUCT Error: Unknown input format     *
 *                                         specified                *
 *              - Message: DSN8DUCT Error: Inputted time must indi- *
 *                                         cate either AM or PM     *
 *              - Message: DSN8DUCT Error: Hour not in expected range *
 *                                         of 1-12                  *
 *              - Message: DSN8DUCT Error: Hour not in expected range *
 *                                         of 0-23                  *
 *              - Message: DSN8DUCT Error: Minute must be 2 numerics *
 *                                         between 00 and 59        *
 *                                         to input format          *
 *              - Message: DSN8DUCT Error: Second must be 2 numerics *
 *                                         between 00 and 59        *
 *                                         to input format          *
 *                                                                  *
 *              Return Code: SQLSTATE = 38603                       *
 *              - Message: DSN8DUCT Error: Unknown output format    *
 *                                         specified                *
 *                                                                  *
 *    External References:                                          *
 *             - Routines/Services: None                            *
 *             - Data areas       : None                            *
 *             - Control blocks   : None                            *
 *                                                                  *
 *                                                                  *
 *  Pseudocode:                                                     *
 *   DSN8DUCT:                                                      *
 *   - Issue sqlstate 38601 and a diagnostic message if no input time *
 *     was provided.                                                *
 *   - Issue sqlstate 38601 and a diagnostic message if no input for- *
 *     mat was provided.                                            *
 *   - Issue sqlstate 38601 and a diagnostic message if no output   *
 *     format was provided.                                         *
 *   - Call decontime to deconstruct the input time into hour, minute,*
 *     and, if either, second and AM/PM indicator, according to the *
 *     input format.                                                *
 *   - Call recontime to create an output time from the hour, minute, *
 *     and, if either, second and AM/PM indicator, according to the *
 *     output format.                                               *
 *   - If no errors, unset null indicators, and return SQLSTATE 00000 *
 *     else set null indicator and return null time out.            *
 *   End DSN8DUCT                                                   *
 *                                                                  *
 *   deconTime                                                      *
 *   - Parse hour, minute, and, if either, second and AM/PM indicator *
 *     from the input time by breaking on delimiters (: and .).     *
 *   - Use the input format to determine sequence of time components. *
 *     - if format invalid, issue SQLSTATE 38602 and a diag. message *
 *   - Call checkHour to validate the hour component and to standard- *
 *     ize it (if required) from a 12-hour clock to a 24-hour clock. *
 *     - if not valid hour, issue SQLSTATE 38602 and a diag. message *
 *   - Call checkMinute to validate the minute component            *
 *     - if not valid minute, issue SQLSTATE 38602 and a diag. msg.  *
 *   - If applicable, call checkSecond to validate the second comp.  *
 *     - if not valid second, issue SQLSTATE 38602 and a diag. msg.  *
 *   - If applicable, call checkAMPMindicator to validate the AM/PM  *
```

```
*     indicator                                                      *
*     - if not valid indicator, issue SQLSTATE 38602 and a diag. msg.*
*   End deconTime                                                    *
*                                                                    *
*   reconTime                                                        *
*   - Use the output format to edit the time components              *
*     - call set12HrClock to convert the hours component from 24-    *
*       hour clock format, as appropriate                            *
*     - call remove0prefix to strip the leading 0 from the hour com- *
*       ponent, if appropriate                                       *
*     - if output format is invalid, issue SQLSTATE 38603 and a      *
*       diagnostic message                                           *
*   - Call buildTime to create the output time from the edited time  *
*     components                                                     *
*   End reconTime                                                    *
*                                                                    *
*   buildTime                                                        *
*   - Generate the time out by concatenating the time componentents  *
*     (hour, minute, and, optionally, second and/or AM/PM indicator) *
*     with intervening delimiters (: or .).                          *
*   End buildTime                                                    *
*                                                                    *
*   checkHour                                                        *
*   - Verify that the hour component of the input time is:           *
*     - in the range 01 - 12 if the input format carries an AM/PM    *
*       indicator                                                    *
*       - call set24HrClock to standardize the hour to a 24-hour     *
*         clock format.                                              *
*     - in the range 00 - 23 if the input format does not carry an   *
*       AM/PM indicator                                              *
*   - If not valid, set error flag and return null value for hour    *
*   End checkHour                                                    *
*                                                                    *
*   checkMinute                                                      *
*   - Verify the minute component is 2 digits ranging from 00 - 59.  *
*   - If not valid, set error flag and return null value for minute  *
*   End checkMinute                                                  *
*                                                                    *
*   checkSecond                                                      *
*   - Verify the second component is 2 digits ranging from 00 - 59.  *
*   - If not valid, set error flag and return null value for second  *
*   End checkSecond                                                  *
*                                                                    *
*   checkAMPMindicator                                               *
*   - Verify the AM/PM indicator is either "AM" or "PM"              *
*   - If not valid, set error flag and return null value for         *
*     AM/PM indicator                                                *
*   End checkAMPMindicator                                           *
*                                                                    *
*   set12HrClock                                                     *
*   - Convert a 24-hour clock hour to a 12-hour clock hour           *
*   End set12HrClock                                                 *
*                                                                    *
*   set24HrClock                                                     *
*   - Convert a 12-hour clock hour to a 24-hour clock hour           *
*   End set24HrClock                                                 *
*                                                                    *
*   add0prefix                                                       *
*   - prepend an hour with a leading 0 if it is less than 10         *
*   End add0prefix                                                   *
*                                                                    *
*   remove0prefix                                                    *
*   - strip leading zero from an hour if it is less than 10          *
*   End remove0prefix                                                *
*                                                                    *
*                                                                    *
*********************************************************************/

#pragma linkage(DSN8DUCT,fetchable)

/********************** C library definitions ***********************/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <ctype.h>

/***************************** Equates ******************************/
#define     NULLCHAR     '\0'         /* Null character            */
#define     MATCH          0          /* Comparison status: Equal   */
#define     NOT_OK         1          /* Run status indicator: Error*/
#define     OK             0          /* Run status indicator: Good */
```

```
/************************* Global constants **************************/
char        *clock12[24]             /* map of 12-hour clock       */
            = { "12", "01", "02", "03", "04", "05",
                "06", "07", "08", "09", "10", "11" };

char        *clock24[24]             /* map of 24-hour clock       */
            = { "00", "01", "02", "03", "04", "05",
                "06", "07", "08", "09", "10", "11",
                "12", "13", "14", "15", "16", "17",
                "18", "19", "20", "21", "22", "23" };

char        *char0 = "0";            /* string with character "0"  */


/*********************** DSN8DUCT functions ************************/

void DSN8DUCT                        /* main routine               */
( char        *timeIn,               /* in: time to be converted   */
  char        *formatIn,             /* in: format of timeIn       */
  char        *formatOut,            /* in: format for timeOut     */
  char        *timeOut,              /* out: reformatted time      */
  short int   *nullTimeIn,           /* in: indic var for timeIn   */
  short int   *nullFormatIn,         /* in: indic var for formatIn */
  short int   *nullFormatOut,        /* in: indic var, formatOut   */
  short int   *nullTimeOut,          /* out: indic var for timeOut */
  char        *sqlstate,             /* out: SQLSTATE              */
  char        *fnName,               /* in: family name of function*/
  char        *specificName,         /* in: specific name of func  */
  char        *message               /* out: diagnostic message    */
);

int deconTime                        /* get hr,min,sec from timeIn */
( char        *hour,                 /* out: hour component        */
  char        *minute,               /* out: minute component      */
  char        *second,               /* out: second component      */
  char        *message,              /* out: diagnostic message    */
  char        *sqlstate,             /* out: SQLSTATE              */
  char        *timeIn,               /* in: time to deconstruct    */
  char        *formatIn              /* in: format of timeIn       */
);

int reconTime                        /* get timeOut from hr,min,sec*/
( char        *timeOut,              /* out: reformatted time      */
  char        *message,              /* out: diagnostic message    */
  char        *sqlstate,             /* out: SQLSTATE              */
  char        *hour,                 /* in: hour component         */
  char        *minute,               /* in: minute component       */
  char        *second,               /* in: second component       */
  char        *formatOut             /* in: format for timeOut     */
);

void buildTime                       /* bld timeOut from hr,min,sec*/
( char        *timeOut,              /* out: reformatted time      */
  char        *hour,                 /* in: hour component         */
  char        *minute,               /* in: minute component       */
  char        *second,               /* in: second component       */
  char        *delim,                /* in: delimiter              */
  char        *AMPMind               /* in: AM/PM indic. (if any)  */
);

int checkHour                        /* verify/standardize hourIn  */
( char        *hourOut,              /* out: hour (24 hour clock)  */
  char        *hourIn,               /* in: hour (12- or 24-hr clk)*/
  char        *AMPMind               /* in: AM/PM indicator        */
);

int checkMinute                      /* verify minute from timeIn  */
( char        *minOut,               /* out: minute, validated     */
  char        *minIn                 /* in: minute, unvalidated    */
);

int checkSecond                      /* verify second from timeIn  */
( char        *secOut,               /* out: second, validated     */
  char        *secIn                 /* in: second, unvalidated    */
);

int checkAMPMindicator               /* verify AM/PM ind. of timeIn*/
( char        *indOut,               /* out: AM/PM indic, validated*/
  char        *indIn                 /* in: AM/PM ind, unvalidated */
);

int set12HrClock                     /* hour to 12-hr clock format */
```

```
( char        *hour,                  /* in/out: hour               */
  char        *AMPMind                /* out: AM/PM indicator       */
);

int set24HrClock                      /* hour to 24-hr clock format */
( char        *hour,                  /* in/out: hour               */
  char        *AMPMind                /* in: AM/PM indicator        */
);

void add0Pref                         /* add leading zero to string */
( char        *str3                   /* in/out: string to prefix   */
);

void remove0prefix                    /* strip leading zeroes       */
( char        *string                 /* in/out: character string   */
);

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
void DSN8DUCT
( char        *timeIn,                /* in: time to be converted   */
  char        *formatIn,              /* in: format of timeIn       */
  char        *formatOut,             /* in: format for timeOut     */
  char        *timeOut,               /* out: reformatted time      */
  short int   *nullTimeIn,            /* in: indic var for timeIn   */
  short int   *nullFormatIn,          /* in: indic var for formatIn */
  short int   *nullFormatOut,         /* in: indic var, formatOut   */
  short int   *nullTimeOut,           /* out: indic var for timeOut */
  char        *sqlstate,              /* out: SQLSTATE              */
  char        *fnName,                /* in: family name of function*/
  char        *specificName,          /* in: specific name of func  */
  char        *message                /* out: diagnostic message    */
)
/********************************************************************
*                                                                   *
* Assumptions:                                                      *
* - *timeIn        points to a char[12], null-terminated string     *
* - *formatIn,     points to a char[15], null-terminated string     *
* - *formatOut     points to a char[15], null-terminated string     *
* - *timeOut       points to a char[12], null-terminated string     *
* - *nullTimeIn    points to a short integer                        *
* - *nullFormatIn  points to a short integer                        *
* - *nullFormatOut points to a short integer                        *
* - *nullTimeOut   points to a short integer                        *
* - *sqlstate      points to a char[06], null-terminated string     *
* - *fnName        points to a char[138], null-terminated string    *
* - *specificName  points to a char[129], null-terminated string    *
* - *message       points to a char[70], null-terminated string     *
*******************************************************************/
{
  /************************ Local variables ************************/
  short int   i;                      /* loop control               */
  char        hour[3];                /* gets hour from timeIn      */
  char        minute[3];              /* gets minute from timeIn    */
  char        second[3];              /* gets second from timeIn    */

  short int   status = OK;            /* DSN8DUCT run status        */


  /******************************************************************
  * Verify that an input time, its current format, and its new format*
  * have been passed in.                                            *
  ******************************************************************/
  if( *nullTimeIn || ( strlen( timeIn ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCT Error: No input time entered" );
      strcpy( sqlstate, "38601" );
    }
  else if( *nullFormatIn || ( strlen( formatIn ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCT Error: No input format entered" );
      strcpy( sqlstate, "38601" );
    }
  else if( *nullFormatOut || ( strlen( formatOut ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
```

```
              "DSN8DUCT Error: No output format entered" );
      strcpy( sqlstate, "38601" );
    }

  /*******************************************************************
  * Use formatIn to deconstruct timeIn into hour and minute and, if  *
  * applicable, second.                                              *
  *******************************************************************/
  if( status == OK )
    status = deconTime( hour, minute, second, message, sqlstate,
                        timeIn, formatIn );

  /*******************************************************************
  * Use formatOut to reconstruct timeOut from ours and minute and,   *
  * if applicable, second.                                           *
  *******************************************************************/
  if( status == OK )
    status = reconTime( timeOut, message, sqlstate,
                        hour, minute, second, formatOut );

  /*******************************************************************
  * If conversion was successful, clear the message buffer and sql-  *
  * state, and unset the SQL null indicator for timeOut.             *
  *******************************************************************/
  if( status == OK )
    {
      *nullTimeOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /*******************************************************************
  * If errors occurred, clear the timeOut buffer and set the SQL null*
  * indicator.  A diagnostic message and the SQLSTATE have been set  *
  * where the error was detected.                                    *
  *******************************************************************/
  else
    {
      timeOut[0] = NULLCHAR;
      *nullTimeOut = -1;
    }

  return;

} /* end DSN8DUCT */


/*********************************************************************/
/**************************** functions ******************************/
/*********************************************************************/
int deconTime
( char        *hour,                  /* out: hour component       */
  char        *minute,                /* out: minute component     */
  char        *second,                /* out: second component     */
  char        *message,               /* out: diagnostic message   */
  char        *sqlstate,              /* out: SQLSTATE             */
  char        *timeIn,                /* in: time to deconstruct   */
  char        *formatIn               /* in: format of timeIn      */
)
/*********************************************************************
* Deconstructs *timeIn into *hour and *minute and, if applicable,    *
* *second.   The deconstruction is done according to the value in    *
* formatIn.  Returns OK if deconstruction succeeds, otherwise places *
* diagnostic text in *message and returns NOT_OK.                    *
*********************************************************************/
{
  char        AMPMind[3];             /* AM/PM indicator           */

  char        workTimeIn[12];         /* work copy of timeIn       */
  char        *token;                 /* string ptr for token parser*/
  char        tok1[3];                /* holds 1st time component  */
  char        tok2[3];                /* holds 2nd time component  */
  char        tok3[3];                /* holds 3rd time component  */
  char        tok4[3];                /* holds 4th time component  */

  short int   func_status = OK;       /* function status indicator */
  short int   fmtStatus = OK;         /* indicates if format  is OK */
  short int   hrStatus = OK;          /* indicates if hour    is OK */
  short int   minStatus = OK;         /* indicates if minute  is OK */
  short int   secStatus = OK;         /* indicates if second  is OK */
  short int   indStatus = OK;         /* indicates if AMPMind is OK */

  /*******************************************************************
```

```
* Use C strtok function to parse the hour and minute from timeIn   *
*******************************************************************/
strcpy( workTimeIn,timeIn );
token = strtok( workTimeIn,".: " );
strcpy( tok1,token );
token = strtok( NULL,".: " );
strcpy( tok2,token );

/*******************************************************************
* Parse second, if any, and AM/PM indicator, if any, from timeIn   *
*******************************************************************/
token = strtok( NULL,".: " );
strcpy( tok3,token );
token = strtok( NULL," " );
strcpy( tok4,token );

/*******************************************************************
* Use formatIn to check and set hour, minute, etc.                 *
*******************************************************************/
if( ( strcmp( formatIn,"H:MM AM/PM" ) == MATCH )
 || ( strcmp( formatIn,"HH:MM AM/PM" ) == MATCH ) )
  {
    indStatus = checkAMPMindicator( AMPMind,tok3 );
    hrStatus = checkHour( hour,tok1,AMPMind );
    minStatus = checkMinute( minute,tok2 );
    strcpy( second,"00" );
  }
else if( strcmp( formatIn,"HH:MM:SS AM/PM" ) == MATCH )
  {
    indStatus = checkAMPMindicator( AMPMind,tok4 );
    hrStatus = checkHour( hour,tok1,AMPMind );
    minStatus = checkMinute( minute,tok2 );
    secStatus = checkSecond( second,tok3 );
  }
else if( ( strcmp( formatIn,"HH:MM:SS" ) == MATCH )
      || ( strcmp( formatIn,"H.MM.SS" ) == MATCH )
      || ( strcmp( formatIn,"HH.MM.SS" ) == MATCH ) )
  {
    hrStatus = checkHour( hour,tok1,"" );
    minStatus = checkMinute( minute,tok2 );
    secStatus = checkSecond( second,tok3 );
  }
else if( ( strcmp( formatIn,"H.MM" ) == MATCH )
      || ( strcmp( formatIn,"HH.MM" ) == MATCH ) )
  {
    hrStatus = checkHour( hour,tok1,"" );
    minStatus = checkMinute( minute,tok2 );
    strcpy( second,"00" );
  }
else
  fmtStatus = NOT_OK;

/*******************************************************************
* set up error handling                                            *
*******************************************************************/
func_status = NOT_OK;
strcpy( message,"DSN8DUCT Error: " );
strcpy( sqlstate, "38602" );

/*******************************************************************
* if error detected, issue diagnostic message and return NOT_OK    *
*******************************************************************/
if( fmtStatus != OK )
  strcat( message,"Unknown input format specified" );

else if( indStatus != OK )
  strcat( message,"Inputted time must indicate either AM or PM" );

else if( hrStatus != OK )
  if( strcmp( AMPMind,"AM" ) == MATCH
   || strcmp( AMPMind,"PM" ) == MATCH )
    strcat( message,"Hour not in expected range of 1-12" );
  else
    strcat( message,"Hour not in expected range of 0-23" );

else if( minStatus != OK )
  strcat( message,"minute must be 2 numerics between 00 and 59" );

else if( secStatus != OK )
  strcat( message,"second must be 2 numerics between 00 and 59" );

/*******************************************************************
```

```
  * if no error detected, clear message and sqlstate and return OK   *
  *******************************************************************/
  else
    {
      *message = NULLCHAR;
      func_status = OK;
      strcpy( sqlstate, "00000" );
    }

  return( func_status );

} /* end deconTime */


int reconTime
( char       *timeOut,             /* out: reformatted time      */
  char       *message,             /* out: diagnostic message    */
  char       *sqlstate,            /* out: SQLSTATE              */
  char       *hour,                /* in: hour component         */
  char       *minute,              /* in: minute component       */
  char       *second,              /* in: second component       */
  char       *formatOut            /* in: format for timeOut     */
)
/*********************************************************************
* Reconstructs *timeOut from *hour and *minute and, if applicable,   *
* *second.   The reconstruction is done according to the value in    *
* formatOut.  Returns OK if reconstruction succeeds, otherwise       *
* places diagnostic text in *message and returns NOT_OK.             *
*********************************************************************/
{
  short int   func_status = OK;      /* function status indicator  */
  char        AMPMind[3];            /* AM/PM indicator            */


  /*******************************************************************
  * Use formatOut to reformat time from hour, minute, second         *
  *******************************************************************/
  if( strcmp( formatOut,"H:MM AM/PM" ) == MATCH )
    {
      set12HrClock( hour,AMPMind );
      remove0prefix( hour );
      buildTime( timeOut, hour, minute, "", ":", AMPMind );
    }
  else if( strcmp( formatOut,"HH:MM AM/PM" ) == MATCH )
    {
      set12HrClock( hour,AMPMind );
      buildTime( timeOut, hour, minute, "", ":", AMPMind );
    }
  else if( strcmp( formatOut,"HH:MM:SS AM/PM" ) == MATCH )
    {
      set12HrClock( hour,AMPMind );
      buildTime( timeOut, hour, minute, second, ":", AMPMind );
    }
  else if( strcmp( formatOut,"HH:MM:SS" ) == MATCH )
    {
      buildTime( timeOut, hour, minute, second, ":", "" );
    }
  else if( strcmp( formatOut,"H.MM.SS" ) == MATCH )
    {
      remove0prefix( hour );
      buildTime( timeOut, hour, minute, second, ".", "" );
    }
  else if( strcmp( formatOut,"HH.MM.SS" ) == MATCH )
    {
      buildTime( timeOut, hour, minute, second, ".", "" );
    }
  else if( strcmp( formatOut,"H.MM" ) == MATCH )
    {
      remove0prefix( hour );
      buildTime( timeOut, hour, minute, "", ".", "" );
    }
  else if( strcmp( formatOut,"HH.MM" ) == MATCH )
    {
      buildTime( timeOut, hour, minute, "", ".", "" );
    }
  else
    {
      func_status = NOT_OK;
      strcpy( message,"DSN8DUCT Error: " );
      strcat( message,"Unknown output format specified" );
      strcpy( sqlstate, "38603" );
    }
```

```

  return( func_status );

} /* end reconTime */


void buildTime
( char        *timeOut,              /* out: reformatted time      */
  char        *hour,                 /* in: hour component         */
  char        *minute,               /* in: minute component       */
  char        *second,               /* in: second component       */
  char        *delim,                /* in: delimiter              */
  char        *AMPMind               /* in: AM/PM indic. (if any)  */
)
/*******************************************************************
* Builds *timeOut from *hour, *minute, and (if specified) *second,   *
* separated by the value in *delim and, if specified, suffixed by the*
* value in *AMPMind.                                                 *
*******************************************************************/
{
  /*****************************************************************
  * Build timeOut from incoming time components                     *
  *****************************************************************/
  strcpy( timeOut,hour );            /* Start with hour ...        */
  strcat( timeOut,delim );           /* append the delimiter       */
  strcat( timeOut,minute );          /* append minute              */
  if( strlen(second) > 0 )           /* and, if second specified,  */
    {
      strcat( timeOut,delim );       /* ..append the delimiter     */
      strcat( timeOut,second );      /* ..append second            */
    }
  if( strlen(AMPMind) > 0 )          /* and, if AM/PM ind. spec'd  */
    {
      strcat( timeOut," " );         /* ..append separator blank   */
      strcat( timeOut,AMPMind );     /* ..append AM/PM indicator   */
    }

} /* end buildTime */


int checkHour
( char        *hourOut,              /* out: hour (24-hour clock)  */
  char        *hourIn,               /* in: hour (12- or 24-hr clk)*/
  char        *AMPMind               /* in: AM/PM indicator        */
)
/*******************************************************************
* Verifies that *hourIn meets one of these criteria:                *
* - if *AMPMind is "AM" or "PM", *hourIn ranges from "01" to "12"    *
* - otherwise, *hourIn ranges from "0" to "23"                       *
*                                                                    *
* If the appropriate criterion is met, *hourOut is assigned as       *
* follows:                                                           *
* - if *AMPMind is "AM" or "PM", *hourOut is assigned the 24-hour    *
*   clock equivalent of *hourIn.                                     *
* - otherwise, *hourIn is copied to *hourOut                         *
* and checkHour returns OK.                                          *
*                                                                    *
* If the appropriate critierion is not met, *hourOut is assigned     *
* NULLCHAR, and checkHour returns NOT_OK.                            *
*******************************************************************/
{
  short int   i;                     /* loop control               */
  short int   func_status = OK;      /* function status indicator  */

  /*****************************************************************
  * add leading 0, if needed, to *hourIn                            *
  *****************************************************************/
  add0Pref( hourIn );

  /*****************************************************************
  * if AMPMind is AM or PM, convert *hourIn to 24-clock format      *
  *****************************************************************/
  if( strcmp( AMPMind,"AM" ) == MATCH
   || strcmp( AMPMind,"PM" ) == MATCH )
    func_status = set24HrClock( hourIn,AMPMind );

  /*****************************************************************
  * if AMPMind not "AM" or "PM", verify hourIn ranges from 00 to 23 *
  *****************************************************************/
  else
    {
      for( i=0; i<24 && strcmp( hourIn,clock24[i] ) != MATCH; i++ );
```

```
      if( i >= 24 )                   /* if hourIn < 00 & > 23      */
        func_status = NOT_OK;         /* ..set error flag           */
    }

  /*******************************************************************
  * if *hourIn is valid, copy it to *hourOut else set *hourOut to    *
  * NULLCHAR                                                         *
  *******************************************************************/
  if( func_status == OK )
    strcpy( hourOut,hourIn );
  else
    hourOut[0] = NULLCHAR;

  return( func_status );
} /* end checkHour */


int checkMinute
( char        *minOut,                /* out: minute, validated     */
  char        *minIn                  /* in: minute, unvalidated    */
)
/*********************************************************************
* Verifies that *minIn is 2 bytes of numeric characters between "00" *
* and "59".                                                          *
*                                                                    *
* If so, minIn is copied to minOut and checkMinute returns OK.       *
* If not, NULLCHAR is copied to minOut and checkMinute returns       *
* NOT_OK.                                                            *
*********************************************************************/
{
  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * verify that *minIn is 2 numeric characters between "00" and "59" *
  *******************************************************************/
  if( strlen( minIn ) != 2 )
    func_status = NOT_OK;
  else if( isdigit(minIn[0]) == MATCH || isdigit(minIn[1]) == MATCH )
    func_status = NOT_OK;
  else if( strcmp( minIn,"00" ) < 0 || strcmp( minIn,"59" ) > 0 )
    func_status = NOT_OK;

  /*******************************************************************
  * if minIn is valid, assign it to minOut                           *
  *******************************************************************/
  if( func_status == OK )
    strcpy( minOut,minIn );
  else
    minOut[0] = NULLCHAR;

  return( func_status );
} /* end checkMinute */


int checkSecond
( char        *secOut,                /* out: second, validated     */
  char        *secIn                  /* in: second, unvalidated    */
)
/*********************************************************************
* Verifies that *secIn is 2 bytes of numeric characters between "00" *
* and "59".                                                          *
*                                                                    *
* If so, secIn is copied to secOut and checkSecond returns OK.       *
* If not, NULLCHAR is copied to secOut and checkSecond returns       *
* NOT_OK.                                                            *
*********************************************************************/
{
  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * verify that *secIn is 2 numeric characters between "00" and "59" *
  *******************************************************************/
  if( strlen( secIn ) != 2 )
    func_status = NOT_OK;
  else if( isdigit(secIn[0]) == MATCH || isdigit(secIn[1]) == MATCH )
    func_status = NOT_OK;
  else if( strcmp( secIn,"00" ) < 0 || strcmp( secIn,"59" ) > 0 )
    func_status = NOT_OK;

  /*******************************************************************
```

```
  * if secIn is valid, assign it to secOut                            *
  ********************************************************************/
  if( func_status == OK )
    strcpy( secOut,secIn );
  else
    secOut[0] = NULLCHAR;

  return( func_status );
} /* end checkSecond */


int checkAMPMindicator
( char        *indOut,                /* out: AM/PM indic, validated*/
  char        *indIn                  /* in: AM/PM ind, unvalidated */
)
/*********************************************************************
* Verifies that *indIn is 2 bytes, containing either "AM" or "PM".   *
*                                                                    *
* If so, indIn is copied to indOut and checkAMPMindicator returns    *
* OK.                                                                *
* If not, NULLCHAR is copied to indOut and checkAMPMindicator re-    *
* turns NOT_OK.                                                      *
*********************************************************************/
{
  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * verify that *indIn is 2 bytes containing either "AM" or "PM"     *
  *******************************************************************/
  if( strlen( indIn ) != 2 )
    func_status = NOT_OK;
  else if( strcmp(indIn,"AM") != MATCH && strcmp(indIn,"PM") != MATCH )
    func_status = NOT_OK;

  /*******************************************************************
  * if indIn is valid, assign it to indOut                           *
  *******************************************************************/
  if( func_status == OK )
    strcpy( indOut,indIn );
  else
    indOut[0] = NULLCHAR;

  return( func_status );
} /* end checkAMPMindicator */


int set12HrClock
( char        *hour,                  /* in/out: hour               */
  char        *AMPMind                /* out: AM/PM indicator       */
)
/*********************************************************************
* Changes *hour from 24-hour clock format to 12-hour clock format.   *
*                                                                    *
* If the incoming value for *hour is:                                *
* - between "00" and "11", *hour is assigned the 12-hour clock       *
*   equivalent ("12" - "11"), *AMPMind is assigned "AM", and         *
*   set12HrClock returns OK.                                         *
* - between "12" and "23", *hour is assigned the 12-hour clock       *
*   equivalent ("12" - "11"), *AMPMind is assigned "PM", and         *
*   set12HrClock returns OK.                                         *
* - any other value, *hour is unchanged, *AMPMind is assigned        *
*   NULLCHAR, and set12HrClock returns NOT_OK.                       *
*********************************************************************/
{
  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*******************************************************************
  * locate *hour in the 24-hour clock map                            *
  *******************************************************************/
  for( i=0; i<24 && strcmp( hour,clock24[i] ) != MATCH; i++ );

  /*******************************************************************
  * assign *hour its 12-hour clock equivalent                        *
  *******************************************************************/
  if( i < 12 )                        /* if hour betw/ "00" & "11"  */
    {
      strcpy( hour,clock12[i] );      /* ..set hour in 12-hour fmt  */
      strcpy( AMPMind,"AM" );         /* ..set AM/PM indic to AM    */
    }
  else if( i < 24 )                   /* if hour betw/ "12" & "23"  */
```

```
    {
      strcpy( hour,clock12[i-12] );   /* ..set hour in 12-hour fmt  */
      strcpy( AMPMind,"PM" );         /* ..set AM/PM indic to PM    */
    }
  else                                /* otherwise ..               */
    {
      func_status = NOT_OK;           /* ..set error flag           */
      AMPMind[0] = NULLCHAR;          /* ..null out AM/PM indicator */
    }

  return( func_status );              /* return function status     */
} /* end set12HrClock */


int set24HrClock
( char        *hour,                  /* in/out: hour               */
  char        *AMPMind                /* in: AM/PM indicator        */
)
/*******************************************************************
* Changes *hour from 12-hour clock format to 24-hour clock format.  *
*                                                                   *
* If the incoming value for *hour is not between "01" and "12",     *
* then *hour is unchanged and set24HrClock returns NOT_OK.          *
*                                                                   *
* Otherwise:                                                        *
* - if *AMPMind is "AM", then *hour is assigned the 24-hour equiva- *
*   lent of morning hour ("00"-"11") and set24HrClock returns OK.   *
* - else *hour is assigned the 24-hour equivalent of afternoon      *
*   hour ("12"-"23") and set24HrClock returns OK.                   *
*******************************************************************/
{
  short int   i;                      /* loop control               */
  short int   func_status = OK;       /* function status indicator  */

  /*****************************************************************
  * locate *hour in the 12-hour clock map                           *
  *****************************************************************/
  for( i=0; i<12 && strcmp( hour,clock12[i] ) != MATCH; i++ );

  /*****************************************************************
  * assign *hour its 24-hour clock equivalent                       *
  *****************************************************************/
  if( i > 11 )                        /* if hour not betw/ 01 & 12  */
    func_status = NOT_OK;             /* ..set error flag           */
  else if(strcmp(AMPMind,"AM")==MATCH)/* else if betw/ 12 AM - 11 AM*/
    strcpy( hour,clock24[i] );        /* ..set hour in 12-hour fmt  */
  else                                /* else betw/ 12 PM - 11 PM   */
    strcpy( hour,clock24[i+12] );     /* ..set hour in 12-hour fmt  */

  return( func_status );              /* return function status     */
} /* end set24HrClock */


void add0Pref
( char        *str3                   /* in/out: string to prefix   */
)
/*******************************************************************
* Prefixes *str3 with a leading 0 if it is only 1 byte long         *
*******************************************************************/
{
  /*****************************************************************
  * if str3 is just 1 byte long, prefix it with a "0"               *
  *****************************************************************/
  if( strlen( str3 ) == 1 )
    {
      str3[1] = str3[0];              /* Right-shift *str3 1 byte   */
      str3[0] = *char0;               /* Prefix it with "0"         */
      str3[2] = NULLCHAR;             /* And terminate it           */
    }
} /* end add0Pref */


void remove0prefix
( char        *string                 /* in/out: character string   */
)
/*******************************************************************
* Eliminates all leading zeroes from *str3.  Leaves a single zero in *
* the first byte of *str3 if *str3 is all zeroes.                   *
*******************************************************************/
{
  short int   i = 0;                  /* Loop control               */
  short int   j = 0;                  /* Loop control               */
```

```

  /*****************************************************************
  * if leading zero in first byte, skip up to first non-zero        *
  *****************************************************************/
  if( string[0] == '0' )
    for( i=0; string[i] == '0'; i++ );

  /*****************************************************************
  * if at end of string, it was all zeroes: put zero in 1st byte    *
  *****************************************************************/
  if( string[i] == '\0' )
    strcpy( string,"0" );
  /*****************************************************************
  * otherwise, left-shift non-zero chars and terminate string       *
  *****************************************************************/
  else
    {
      for( j=0; string[i] != NULLCHAR; j++ )
        string[j] = string[i++];
      string[j] = NULLCHAR;
    }
} /* end remove0prefix */
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUCY

Formats a given numeric amount with a specified currency symbol and, if specified, one of the following debit/ credit indicators.

```
/*********************************************************************
* Module name = DSN8DUCY (DB2 sample program)                       *
*                                                                   *
* DESCRIPTIVE NAME = General currency formatter (UDF)               *
*                                                                   *
*                                                                   *
*    LICENSED MATERIALS - PROPERTY OF IBM                           *
*    5675-DB2                                                       *
*    (C) COPYRIGHT 1998, 2000 IBM CORP.  ALL RIGHTS RESERVED.       *
*                                                                   *
*    STATUS = VERSION 7                                             *
*                                                                   *
* Function: Formats a given numeric amount with a specified currency *
*           symbol and, if specified, one of the following debit/   *
*           credit indicators.                                      *
*                                                                   *
*           +/-: Place a hyphen between the currency symbol and the *
*                amount if the amount is less than 0.               *
*           (/): Place a left parenthesis between currency symbol   *
*                and the amount and place a right parenthesis to the *
*                right of the amount if the amount is less than 0.  *
*         CR/DB: Place CR to the right of the amount if it is less  *
*                than 0; otherwise place DB to the right of the     *
*                amount.                                            *
*                                                                   *
*           Example invocations:                                    *
*            EXEC SQL SET :money = CURRENCY( -123,                  *
*                                            "DM" );                *
*            ==> money = DM -123.00                                 *
*                                                                   *
*                                                                   *
*            EXEC SQL SET :money = CURRENCY( -123,                  *
*                                            "DM",                  *
*                                            "(/)" );               *
*            ==> money = DM (123.00)                                *
*                                                                   *
* Notes:                                                            *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher      *
*                                                                   *
*   Restrictions:                                                   *
*                                                                   *
* Module type: C program                                            *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                  *
* Module size: See linkedit output                                  *
*  Attributes: Re-entrant and re-usable                             *
*                                                                   *
```

```
 * Entry Point: DSN8DUCY                                             * 00004700
 *     Purpose: See Function                                         * 00004800
 *     Linkage: DB2SQL                                               * 00004900
 *              Invoked via SQL UDF call                             * 00005000
 *                                                                   * 00005100
 *  Parameters: DSN8DUCY uses the C "main" argument convention of    * 00005200
 *              argv (argument vector) and argc (argument count).    * 00005300
 *                                                                   * 00005400
 *              The location of input and output parameters depends  * 00005500
 *              on whether the CURRENCY UDF is invoked with two      * 00005600
 *              input arguments (amount and currency symbol) or with * 00005700
 *              three input arguments (amount, currency symbol, and  * 00005800
 *              credit/debit indicator).                             * 00005900
 *                                                                   * 00006000
 *              If the CURRENCY UDF is invoked with two arguments    * 00006100
 *              only (an amount and a currency symbol):              * 00006200
 *              - ARGV[0]  = (input) pointer to a char[9], null-     * 00006300
 *                           terminated string having the name of    * 00006400
 *                           this program (DSN8DUCY)                 * 00006500
 *              - ARGV[1]  = (input) pointer to a double word having * 00006600
 *                           the amount to be formatted as currency. * 00006700
 *              - ARGV[2]  = (input) pointer to a char[3], null-     * 00006800
 *                           terminated string having the currency   * 00006900
 *                           symbol.                                 * 00007000
 *              - ARGV[3]  = (output) pointer to a char[20], null-   * 00007100
 *                           terminated string to receive the cur-   * 00007200
 *                           rency result.                           * 00007300
 *              - ARGV[4]  = (input) pointer to a short integer      * 00007400
 *                           having the null indicator for the input * 00007500
 *                           amount                                  * 00007600
 *              - ARGV[5]  = (input) pointer to a short integer      * 00007700
 *                           having the null indicator for the cur-  * 00007800
 *                           rency symbol                            * 00007900
 *              - ARGV[6]  = (output) pointer to a short integer     * 00008000
 *                           having the null indicator for the result * 00008100
 *              - ARGV[7]  = (output) pointer to a char[6], null-    * 00008200
 *                           terminated string to receive the        * 00008300
 *                           SQLSTATE                                * 00008400
 *              - ARGV[8]  = (input) pointer to a char[138], null-   * 00008500
 *                           terminated string having the UDF family * 00008600
 *                           name of the function                    * 00008700
 *              - ARGV[9]  = (input) pointer to a char[129], null-   * 00008800
 *                           terminated string having the UDF        * 00008900
 *                           specific name of the function           * 00009000
 *              - ARGV[10] = (output) pointer to a char[70],         * 00009100
 *                           null- terminated string to receive any  * 00009200
 *                           diagnostic message issued by this       * 00009300
 *                           function                                * 00009400
 *                                                                   * 00009500
 *              If the CURRENCY UDF is invoked with three arguments  * 00009600
 *              (an amount, a currency symbol, and a credit/debit    * 00009700
 *              indicator):                                          * 00009800
 *              - ARGV[0]  = (input) pointer to a char[9], null-     * 00009900
 *                           terminated string having the name of    * 00010000
 *                           this program (DSN8DUCY)                 * 00010100
 *              - ARGV[1]  = (input) pointer to a double word having * 00010200
 *                           the amount to be formatted as currency. * 00010300
 *              - ARGV[2]  = (input) pointer to a char[3], null-     * 00010400
 *                           terminated string having the currency   * 00010500
 *                           symbol.                                 * 00010600
 *              - ARGV[3]  = (input) pointer to a char[6], null-     * 00010700
 *                           terminated string having the credit/    * 00010800
 *                           debit indicator (see under "Function:", * 00010900
 *                           above, for valid credit/debit indicators)* 00011000
 *              - ARGV[4]  = (output) pointer to a char[20], null-   * 00011100
 *                           terminated string to receive the cur-   * 00011200
 *                           rency result.                           * 00011300
 *              - ARGV[5]  = (input) pointer to a short integer      * 00011400
 *                           having the null indicator for the input * 00011500
 *                           amount                                  * 00011600
 *              - ARGV[6]  = (input) pointer to a short integer      * 00011700
 *                           having the null indicator for the cur-  * 00011800
 *                           rency symbol                            * 00011900
 *              - ARGV[7]  = (input) pointer to a short integer      * 00012000
 *                           having the null indicator for the       * 00012100
 *                           credit/debit indicator                  * 00012200
 *              - ARGV[8]  = (output) pointer to a short integer     * 00012300
 *                           having the null indicator for the result * 00012400
 *              - ARGV[9]  = (output) pointer to a char[6], null-    * 00012500
 *                           terminated string to receive the        * 00012600
 *                           SQLSTATE                                * 00012700
 *              - ARGV[10] = (input) pointer to a char[138], null-   * 00012800
```

```
*                           terminated string having the UDF family  *
*                           name of the function                     *
*              - ARGV[11] = (input) pointer to a char[129], null-    *
*                           terminated string having the UDF         *
*                           specific name of the function            *
*              - ARGV[12] = (output) pointer to a char[70],          *
*                           null- terminated string to receive any   *
*                           diagnostic message issued by this        *
*                           function                                 *
*                                                                    *
* Normal Exit: Return Code: SQLSTATE = 00000                         *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code: SQLSTATE = 38601                         *
*              - Message: DSN8DUCY Error: No amount entered          *
*              - Message: DSN8DUCY Error: No currency symbol entered *
*                                                                    *
*              Return Code: SQLSTATE = 38602                         *
*              - Message: DSN8DUCY Error: Error performing           *
*                                         setlocale()                *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: None                              *
*             - Data areas       : None                              *
*             - Control blocks   : None                              *
*                                                                    *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8DUCY:                                                        *
*   - Walk down the argv list, locating the input and output parms   *
*   - Issue sqlstate 38601 and a diagnostic message if no input      *
*     amount was provided.                                           *
*   - Issue sqlstate 38601 and a diagnostic message if no currency   *
*     symbol was provided.                                           *
*   - Call formatAmount to assemble the currency expression from the *
*     input amount and the currency symbol and, optionally, the      *
*     credit/debit indicator.                                        *
*   - If no errors, unset null indicators, and return SQLSTATE 00000 *
*     else set null indicator and return null time out.              *
*   End DSN8DUCY                                                     *
*                                                                    *
*   formatAmount                                                     *
*   - If the amount in is less than zero ...                         *
*     - if a CR/DB indicator of -/+ has been specified, prefix       *
*       the currency expression with a hyphen                        *
*     - else if a CR/DB of (/) has been specified, prefix the curr-  *
*       rency expression with a left parenthesis                     *
*   - Append the currency symbol to the currency expression          *
*     - if the currency symbol is just 1 byte, concatentate a blank  *
*   - Call the C function setlocale() to initialize the C function   *
*     strfmon().                                                     *
*     - if error, issue SQLSTATE 38602 and a diagnostic message      *
*   - Call the C function strfmon() to reformat the input amount     *
*     with the currency symbol.                                      *
*   - If the amount in is less than zero ...                         *
*     - if a CR/DB of (/) has been specified, append a right paren-  *
*       thesis to the currency expression.                           *
*   End formatAmount                                                 *
*                                                                    *
*                                                                    *
*********************************************************************/
/********************** C library definitions ***********************/
#include <localdef.h>
#include <monetary.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***************************** Equates *****************************/
#define     NULLCHAR     '\0'         /* Null character            */

#define     MATCH          0          /* Comparison status: Equal   */
#define     NOT_OK         0          /* Run status indicator: Error*/
#define     OK             1          /* Run status indicator: Good */


/********************** DSN8DUCY functions ************************/
int main                              /* main routine              */
( int         argc,                   /* standard argument count    */
  char        *argv[]                 /* standard argument vector   */
);
```

```
int formatAmount                      /* format amountIn as currency*/
( char        *moneyOut,              /* out: formatted amountIn    */
  char        *message,               /* out: diagnostic message    */
  char        *sqlstate,              /* out: SQLSTATE              */
  double      *amountIn,              /* in: value to be formatted  */
  char        *currSymbol,            /* in: currency symbol        */
  char        *CRDBInd                /* in: credit/debit indicator */
);

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
int main                              /* main routine               */
( int         argc,                   /* standard argument count    */
  char        *argv[]                 /* standard argument vector   */
)
/********************************************************************
*                                                                  *
*******************************************************************/
{

  /************************ local variables ************************/
  short int   minus1 = -1;            /* default null indic setting */

                                      /* vars for argument vector   */
  double      *amountIn;              /* in: value to be formatted  */
  char        currSymbol[3];          /* in: currency symbol        */
  char        CRDBInd[6];             /* in: credit/debit indicator */
  char        *moneyOut;              /* out: formatted amountIn    */
  short int   *niAmountIn;            /* in: indic var, amountIn    */
  short int   *niCurrSymbol;          /* in: indic var, currSymbol  */
  short int   *niCRDBInd;             /* in: indic var, CRDBInd     */
  short int   *niMoneyOut;            /* out: indic var, moneyOut   */
  char        *sqlstate;              /* out: SQLSTATE              */
  char        fnName[138];            /* in: family name of function*/
  char        specificName[129];      /* in: specific name of func  */
  char        *message;               /* out: diagnostic message    */

  short int   status = OK;            /* DSN8DUCY run status        */



  /*****************************************************************
  * Walk down the argv list, locating the input and output parms     *
  ******************************************************************/
  argc--;                            /* convert argc to base 0 index*/

  message = (char *)argv[argc--];    /* out: point to UDF diag msg  */

  strcpy( specificName,              /* in: save UDF specific name  */
          argv[argc--]);

  strcpy( fnName,argv[argc--] );     /* in: save UDF function name  */

  sqlstate = (char *)argv[argc--];   /* out: point to UDF sqlstate  */

  niMoneyOut                         /* out: point to null indicator*/
    = (short int *)argv[argc--];     /*      variable for result    */

  if( argc == 7 )                    /* if 3 input parms passed     */
    niCRDBInd                        /* ..in: point to null indic.  */
      = (short int *)argv[argc--];   /*       var for CR/DB indic.  */
  else                               /* otherwise it wasn't passed  */
    niCRDBInd = &minus1;             /* ..so define it as null      */

  niCurrSymbol                       /* in: point to null indicator */
    = (short int *)argv[argc--];     /*     var for currency symbol */

  niAmountIn                         /* in: point to null indicator */
    = (short int *)argv[argc--];     /*     var for input amount    */

  moneyOut = (char *)argv[argc--];   /* out: point to UDF result    */

  if( argc == 3 )                    /* if 3 input parms passed     */
    strcpy( CRDBInd,                 /* ..in: save object location  */
            argv[argc--] );          /*       name                  */
  else                               /* otherwise it wasn't passed  */
    CRDBInd[0] = NULLCHAR;           /* ..so define it as null      */

  strcpy( currSymbol,argv[argc--] ); /* in: save currency symbol    */

  amountIn = (double *)argv[argc];   /* in: save input amount       */
```

```

  /*******************************************************************
  * Initialize output parms                                          *
  *******************************************************************/
  message[0] = NULLCHAR;
  strcpy( sqlstate,"00000" );
  *niMoneyOut = 0;
  moneyOut[0] = NULLCHAR;

  /*******************************************************************
  * Verify that an amount and a currency symbol have been passed in  *
  *******************************************************************/
  if( *niAmountIn )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCY Error: No amount entered" );
      strcpy( sqlstate, "38601" );
    }
  else if( *niCurrSymbol || ( strlen( currSymbol ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8DUCY Error: No currency symbol entered" );
      strcpy( sqlstate, "38601" );
    }

  /*******************************************************************
  * Format the amount into currency notation                         *
  *******************************************************************/
  if( status == OK )
    status = formatAmount( moneyOut, message, sqlstate,
                           amountIn, currSymbol, CRDBInd );

  /*******************************************************************
  * If formatting was successful, clear the message buffer and sql-  *
  * state, and unset the SQL null indicator for moneyOut.            *
  *******************************************************************/
  if( status == OK )
    {
      *niMoneyOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /*******************************************************************
  * If errors occurred, clear the moneyOut buff and set the SQL null *
  * indicator.  A diagnostic message and the SQLSTATE have been set  *
  * where the error was detected.                                    *
  *******************************************************************/
  else
    {
      moneyOut[0] = NULLCHAR;
      *niMoneyOut = -1;
    }

  return( 0 );
}  /* end main */

/********************************************************************/
/***************************** functions ****************************/
/********************************************************************/
int formatAmount                      /* format amountIn as currency*/
( char        *moneyOut,              /* out: formatted amountIn    */
  char        *message,               /* out: diagnostic message    */
  char        *sqlstate,              /* out: SQLSTATE              */
  double      *amountIn,              /* in: value to be formatted  */
  char        *currSymbol,            /* in: currency symbol        */
  char        *CRDBInd                /* in: credit/debit indicator */
)
/*********************************************************************
* Converts amountIn to a string, including the currency type in      *
* currSymbol and, if specified, the credit/debit indicator in CRDB-  *
* Symbol.  The result is placed in moneyOut.                         *
*                                                                    *
* currSymbol may be any string of characters, up to 2 bytes long.    *
* CRDBInd is used only its value is:                                 *
* - "+/-", indicating that moneyOut and its prefix from currSymbol   *
*   should be prefixed by a hyphen ("-") if amountIn is negative.    *
* - "(/)", indicating that moneyOut should be enclosed, with the     *
*   prefix from currSymbol, in parentheses if amountIn is negative.  *
* - "CR/DB", indicating that moneyOut should be prefixed with DB     *
```

```
 *   if amountIn is negative, otherwise with CR.                      *
 *****************************************************************/
 {
   int         i;                      /* loop control              */
   int         negFlag = 0;            /* negative currency flag    */

   double      amount;                 /* work var for amountIn     */
   char        moneyStr[200];          /* work string for type conv. */

   /*****************************************************************
   * Clear any residual value from moneyOut                         *
   *****************************************************************/
   for( i=0;i<strlen(moneyOut);i++)
     moneyOut[i] = NULLCHAR;

   /*****************************************************************
   * If amountIn is negative, prefix moneyOut with neg curr indicators*
   *****************************************************************/
   amount = *amountIn;
   if( amount < 0 )
     {
       negFlag = 1;
       if( CRDBInd[0] != NULLCHAR )
         {
             amount = -amount;

             if( strcmp( CRDBInd,"+/-" ) == 0 )
               strcpy( moneyOut,"-" );
             else if( strcmp( CRDBInd,"(/)" ) == 0 )
               strcpy( moneyOut,"(" );
         }
     }

   /*****************************************************************
   * Append the currency type (currSymbol) to moneyOut.  If currSymbol*
   * is more than one byte, place a blank between it and the amount   *
   *****************************************************************/
   strcat( moneyOut,currSymbol );
   if( strlen( currSymbol ) > 1 )
     strcat( moneyOut," " );

   /*****************************************************************
   * Set the local for the strfmon function                         *
   *****************************************************************/
   if( setlocale( LC_ALL, "En_US" ) == NULL )
     {
       strcpy( message,
               "DSN8DUCY Error: Error performing setlocale()" );
       strcpy( sqlstate, "38602" );
       return( NOT_OK );
     }

   /*****************************************************************
   * Reformat amount to a string type with thousands grouping       *
   *****************************************************************/
   strfmon( moneyStr,100,"%!n",amount );
   strcat( moneyOut,moneyStr );

   /*****************************************************************
   * If amount < 0, append negative currency indicators, if passed  *
   *****************************************************************/
   if( CRDBInd[0] != NULLCHAR )
     {
       if( negFlag == 1 )
         {
           if( strcmp( CRDBInd,"CR/DB" ) == 0 )
             strcat( moneyOut," DB" );
           else if( strcmp( CRDBInd,"(/)" ) == 0 )
             strcat( moneyOut,")" );
         }
       else
         {
           if( strcmp( CRDBInd,"CR/DB" ) == 0 )
             strcat( moneyOut," CR" );
         }
     }

   return( OK );
 } /* end formatAmount */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUTI

Returns the name or the schema name or the location name of an alias according to the name of the UDF and the number of input parameters passed, as follows.

```
/********************************************************************* 00010000
* Module name = DSN8DUTI (DB2 sample program)                        * 00020000
*                                                                    * 00030000
* DESCRIPTIVE NAME = Resolve a fully-qualified (3 part), partially-  * 00040000
*                    qualified (2 part), or unqualified alias to the * 00050000
*                    name, schema, or location of its base table or  * 00060000
*                    view.                                           * 00070000
*                                                                    * 00080000
*  LICENSED MATERIALS - PROPERTY OF IBM                              * 00090000
*  5675-DB2                                                          * 00100000
*  (C) COPYRIGHT 1997, 2000 IBM CORP.  ALL RIGHTS RESERVED.          * 00110000
*                                                                    * 00150000
*  STATUS = VERSION 7                                                * 00160000
*                                                                    * 00190000
* Function: Returns the name or the schema name or the location name * 00250000
*           of an alias according to the name of the UDF and the     * 00260000
*           number of input parameters passed, as follows:           * 00270000
*                                                                    * 00280000
*           TABLE_NAME( objectname )                                 * 00290000
*             returns the unqualified name of the object found after * 00300000
*             any alias chains have been resolved.  The specified    * 00310000
*             object name, the default schema, and a location name   * 00320000
*             of "%" (for any location) are used as the starting     * 00330000
*             point of the resolution.  If the starting point does   * 00340000
*             not refer to an alias, the unqualified name of the     * 00350000
*             starting point is returned.  The resulting name may be * 00360000
*             of a table, view, or undefined object.                 * 00370000
*                                                                    * 00380000
*           TABLE_NAME( objectname, objectschema )                   * 00390000
*             returns the unqualified name of the object found after * 00400000
*             any alias chains have been resolved.  The specified    * 00410000
*             object name and schema, and a location name of "%"     * 00420000
*             (for any location), are used as the starting point of  * 00430000
*             the resolution.  If the starting point does not refer  * 00440000
*             to an alias, the unqualified name of the starting  * 00450000
*             point is returned. The resulting name may be of a      * 00460000
*             table, view, or undefined object.                      * 00470000
*                                                                    * 00480000
*           TABLE_NAME( objectname, objectschema, objectlocation )   * 00490000
*             returns the unqualified name of the object found after * 00500000
*             any alias chains have been resolved.  The specified    * 00510000
*             object name, schema, and location name are used as the * 00520000
*             starting point of the resolution.  If the starting     * 00530000
*             point does not refer to an alias, the unqualified name * 00540000
*             of the starting point is returned.  The resulting name * 00550000
*             may be of a table, view, or undefined object.          * 00560000
*                                                                    * 00570000
*           TABLE_SCHEMA( objectname ) returns the schema name of    * 00580000
*             the object found after any alias chains have been      * 00590000
*             resolved.  The specified object name, the default      * 00600000
*             schema, and a location name of "%" (for any location)  * 00610000
*             are used as the starting point of the resolution.  If  * 00620000
*             the starting point does not refer to an alias, the     * 00630000
*             schema name of the starting point is returned.  The    * 00640000
*             resulting schema name may be of a table, view, or      * 00650000
*             undefined object.                                      * 00660000
*                                                                    * 00670000
*           TABLE_SCHEMA( objectname, objectschema ) returns the     * 00680000
*             schema name of the object found after any alias chains * 00690000
*             have been resolved.  The specified object name and     * 00700000
*             schema and a location name of "%" (for any location)   * 00710000
*             are used as the starting point of the resolution.  If  * 00720000
*             the starting point does not refer to an alias, the     * 00730000
*             schema name of the starting point is returned.  The    * 00740000
*             resulting schema name may be of a table, view, or      * 00750000
*             undefined object.                                      * 00760000
*                                                                    * 00770000
*           TABLE_SCHEMA( objectname, objectschema, objectlocation ) * 00780000
*             returns the schema name of the object found after any  * 00790000
```

```
*             alias chains have been resolved.  The specified object *
*             name, schema, and location name are used as the        *
*             starting point of the resolution.  If the starting     *
*             point does not refer to an alias, the schema name of   *
*             starting point is returned.  The resulting schema name *
*             may be of a table, view, or undefined object.          *
*                                                                    *
*           TABLE_LOCATION( objectname ) returns the location name   *
*             of the object found after any alias chains have been   *
*             resolved.  The specified object name, the default      *
*             schema, and a location name of "%" (for any location)  *
*             are used as the starting point of the resolution.  If  *
*             the starting point does not refer to an alias, a blank *
*             location name (indicating the current server) is       *
*             returned.  The resulting location name may be of a     *
*             table, view, or undefined object.                      *
*                                                                    *
*           TABLE_LOCATION( objectname, objectschema ) returns the   *
*             location name of the object found after any alias      *
*             chains have been resolved.  The specified object name, *
*             schema, and a location name of "%" (for any location)  *
*             are used as the starting point of the resolution.  If  *
*             the starting point does not refer to an alias, a blank *
*             location name (indicating the current server) is       *
*             returned.  The resulting location name may be of a     *
*             table, view, or undefined object.                      *
*                                                                    *
*           TABLE_LOCATION( objectname,objectschema,objectlocation ) *
*             returns the location name of the object found after    *
*             any alias chains have been resolved.  The specified    *
*             object name, schema, and location name are used as the *
*             starting point of the resolution.  If the starting     *
*             point does not refer to an alias, a blank location     *
*             name (indicating the current server) is returned.  The *
*             resulting location name may be of a table, view, or    *
*             undefined object.                                      *
*                                                                    *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or subsequent release       *
* Module size: See linkedit output                                   *
*  Attributes: Re-entrant and re-usable                              *
*                                                                    *
* Entry Point: CEESTART (Language Environment entry point)           *
*     Purpose: See Function                                          *
*     Linkage: DB2SQL                                                *
*              Invoked via SQL UDF call                              *
*                                                                    *
*  Parameters: DSN8DUTI uses the C "main" argument convention of     *
*              argv (argument vector) and argc (argument count).     *
*                                                                    *
*              The location of input and output parameters depends   *
*              on whether the UDF (TABLE_NAME, TABLE_SCHEMA, or      *
*              TABLE_SCHEMA) is invoked with one, two, or three      *
*              input arguments.                                      *
*                                                                    *
*              If the UDF was invoked with the object name only:     *
*              - ARGV[0]  = (input) pointer to a char[9], null-      *
*                           terminated string having the name of     *
*                           this program (DSN8DUTI)                  *
*              - ARGV[1]  = (input) pointer to a char[19], null-     *
*                           terminated string having the object name *
*                           to be used as the starting point of the  *
*                           alias resolution                         *
*              - ARGV[2]  = (output) pointer to a null-terminated    *
*                           string to receive the result as follows: *
*                           - char[19] for the TABLE_NAME UDF        *
*                           - char[9] for the TABLE_SCHEMA UDF       *
*                           - char[17] for the TABLE_LOCATION UDF    *
*              - ARGV[3]  = (input) pointer to a short integer       *
*                           having the null indicator for the object *
*                           name                                     *
*              - ARGV[4]  = (output) pointer to a short integer      *
*                           having the null indicator for the result *
*              - ARGV[5]  = (output) pointer to a char[6], null-     *
*                           terminated string to receive the         *
*                           SQLSTATE                                 *
*              - ARGV[6]  = (input) pointer to a char[138], null-    *
```

```
 *                          terminated string having the UDF family  * 01620000
 *                          name of the function                     * 01630000
 *             - ARGV[7]  = (input) pointer to a char[129],          * 01645990
 *                          null-terminated                          * 01651980
 *                          string having the UDF specific name of   * 01660000
 *                          the function                             * 01670000
 *             - ARGV[8]  = (output) pointer to a char[70],          * 01680000
 *                          null-terminated string to receive any    * 01690000
 *                          diagnostic message issued by this        * 01700000
 *                          function                                 * 01710000
 *                                                                   * 01720000
 *             If the UDF was invoked with the object name and the   * 01730000
 *             object schema (but not the object location name):     * 01740000
 *             - ARGV[0]  = (input) pointer to a char[9], null-      * 01750000
 *                          terminated string having the name of     * 01760000
 *                          this program (DSN8DUTI)                  * 01770000
 *             - ARGV[1]  = (input) pointer to a char[19], null-     * 01780000
 *                          terminated string having the object name * 01790000
 *                          to be used in conjunction with the       * 01800000
 *                          object schema as the starting point of   * 01810000
 *                          the alias resolution                     * 01820000
 *             - ARGV[2]  = (input) pointer to a char[9], null-      * 01830000
 *                          terminated string having the object      * 01840000
 *                          schema to be in used in conjunction with * 01850000
 *                          the object name as the starting point of * 01860000
 *                          the alias resolution                     * 01870000
 *             - ARGV[3]  = (output) pointer to a null-terminated    * 01880000
 *                          string to receive the result as follows: * 01890000
 *                          - char[19] for the TABLE_NAME UDF        * 01900000
 *                          - char[9] for the TABLE_SCHEMA UDF       * 01910000
 *                          - char[17] for the TABLE_LOCATION UDF    * 01920000
 *             - ARGV[4]  = (input) pointer to a short integer       * 01930000
 *                          having the null indicator for the object * 01940000
 *                          name                                     * 01950000
 *             - ARGV[5]  = (input) pointer to a short integer       * 01960000
 *                          having the null indicator for the object * 01970000
 *                          schema                                   * 01980000
 *             - ARGV[6]  = (output) pointer to a short integer      * 01990000
 *                          having the null indicator for the result * 02000000
 *             - ARGV[7]  = (output) pointer to a char[6], null-     * 02010000
 *                          terminated string to receive the         * 02020000
 *                          SQLSTATE                                 * 02030000
 *             - ARGV[8]  = (input) pointer to a char[138], null-    * 02040000
 *                          terminated string having the UDF family  * 02050000
 *                          name of the function                     * 02060000
 *             - ARGV[9]  = (input) pointer to a char[129],          * 02070000
 *                          null- terminated                         * 02080000
 *                          string having the UDF specific name of   * 02090000
 *                          the function                             * 02100000
 *             - ARGV[10] = (output) pointer to a char[70],          * 02110000
 *                          null- terminated string to receive any   * 02120000
 *                          diagnostic message issued by this        * 02130000
 *                          function                                 * 02140000
 *                                                                   * 02150000
 *             If the UDF was invoked with the object name and the   * 02160000
 *             object schema and the object location name:           * 02170000
 *             - ARGV[0]  = (input) pointer to a char[9], null-      * 02180000
 *                          terminated string having the name of     * 02190000
 *                          this program (DSN8DUTI)                  * 02200000
 *             - ARGV[1]  = (input) pointer to a char[19], null-     * 02210000
 *                          terminated string having the object name * 02220000
 *                          to be used in conjunction with the       * 02230000
 *                          object schema and the object location    * 02240000
 *                          name as the starting point of the alias  * 02250000
 *                          resolution                               * 02260000
 *             - ARGV[2]  = (input) pointer to a char[9], null-      * 02270000
 *                          terminated string having the object      * 02280000
 *                          schema to be in used in conjunction with * 02290000
 *                          the object name and the object location  * 02300000
 *                          name as the starting point of the alias  * 02310000
 *                          resolution                               * 02320000
 *             - ARGV[3]  = (input) pointer to a char[17], null-     * 02330000
 *                          terminated string having the object      * 02340000
 *                          location name to be used in conjunction  * 02350000
 *                          with the object name and the object      * 02360000
 *                          schema as the starting point of the      * 02370000
 *                          alias resolution                         * 02380000
 *             - ARGV[4]  = (output) pointer to a null-terminated    * 02390000
 *                          string to receive the result as follows: * 02400000
 *                          - char[19] for the TABLE_NAME UDF        * 02410000
 *                          - char[9] for the TABLE_SCHEMA UDF       * 02420000
 *                          - char[17] for the TABLE_LOCATION UDF    * 02430000
```

```
*              - ARGV[5]  = (input) pointer to a short integer       *
*                           having the null indicator for the object *
*                           name                                     *
*              - ARGV[6]  = (input) pointer to a short integer       *
*                           having the null indicator for the object *
*                           schema                                   *
*              - ARGV[7]  = (input) pointer to a short integer       *
*                           having the null indicator for the object *
*                           location name                            *
*              - ARGV[8]  = (output) pointer to a short integer      *
*                           having the null indicator for the result *
*              - ARGV[9]  = (output) pointer to a char[6], null-     *
*                           terminated string to receive the         *
*                           SQLSTATE                                 *
*              - ARGV[10] = (input) pointer to a char[138], null-    *
*                           terminated string having the UDF family  *
*                           name of the function                     *
*              - ARGV[11]  = (input) pointer to a char[129],         *
*                           null- terminated                         *
*                           string having the UDF specific name of   *
*                           the function                             *
*              - ARGV[12] = (output) pointer to a char[70],          *
*                           null- terminated string to receive any   *
*                           diagnostic message issued by this        *
*                           function                                 *
*                                                                    *
* Normal Exit: Return Code: SQLSTATE = 00000                         *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code: SQLSTATE = 38601                         *
*              - Message: DSN8DUTI Error: Invocation by unexpected   *
*                                         UDF having specific name   *
*                                         <specific name>            *
*              Return Code: SQLSTATE = 38602                         *
*              - Message: DSN8DUTI Error: Unexpected SQLCODE,        *
*                                         <SQLCODE>, from SQL SELECT *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: None                              *
*             - Data areas       : None                              *
*             - Control blocks   : None                              *
*                                                                    *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8DUTI:                                                        *
*   - Walk down the argv list, locating the input and output parms   *
*   - If no object name passed, return null result                   *
*   - If no object schema passed, assign default schema (current     *
*     SQLID) to object schema                                        *
*   - Concatenate wildcard ("%") to object location name             *
*   - SELECT TBNAME, TBCREATOR, and LOCATION from SYSIBM.SYSTABLES   *
*     where NAME is the object name, CREATOR is the object creator,  *
*     LOCATION is LIKE the object location name, and TYPE is "A" for *
*     alias.                                                         *
*     - if there's a result (SQLCODE = 0) then                       *
*       - if the TABLE_NAME UDF is the invoker, assign the result    *
*         from TBNAME and return                                     *
*       - else if the TABLE_SCHEMA UDF is the invoker, assign the    *
*         result from TBCREATOR and return                           *
*       - else if the TABLE_LOCATION UDF is the invoker, assign the  *
*         result from LOCATION and return                            *
*       - else an unexpected UDF is the invoker so issue SQLSTATE    *
*         38601 and a diagnstic message and return                   *
*     - else if there's no result (SQLCODE = 100) then               *
*       - if the TABLE_NAME UDF is the invoker, assign the result    *
*         from the object name and return                            *
*       - else if the TABLE_SCHEMA UDF is the invoker, assign the    *
*         result from the object schema and return                   *
*       - else if the TABLE_LOCATION UDF is the invoker, remove the  *
*         trailing search wildcard ("%") from and assign the result  *
*         from LOCATION and return                                   *
*       - else an unexpected UDF is the invoker so issue SQLSTATE    *
*         38601 and a diagnstic message and return                   *
*     - else there's an unexpected SQLCODE so issue SQLSTATE 38602   *
*       and a diagnostic message and return                          *
*   End DSN8DUTI                                                     *
*                                                                    *
*                                                                    *
*********************************************************************/

/********************** C library definitions **********************/
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

/***************************** Equates *****************************/
#define     NULLCHAR     '\0'         /* Null character              */

#define     MATCH          0          /* Comparison status: Equal    */
#define     NOT_OK         0          /* Run status indicator: Error*/
#define     OK             1          /* Run status indicator: Good */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ***********************/
EXEC SQL BEGIN DECLARE SECTION;
  char        hvObjName[19];          /* host var for object name    */
  short int   *niObjName;             /* indic var for hvObjName     */
  char        hvObjSchema[9];         /* host var for obj schema     */
  short int   *niObjSchema;           /* indic var for hvObjSchema   */
  char        hvObjLocation[18];      /* host var for obj location   */
  short int   *niObjLocation;         /* indic var for hvObjLocation*/
  char        hvLOCATION[17];         /* host var for LOCATION col   */
  char        hvTBCREATOR[9];         /* host var for TBCREATOR col */
  char        hvTBNAME[19];           /* host var for TBNAME column */
EXEC SQL END DECLARE SECTION;


int main( int argc, char *argv[] )
{
  /************************ local variables ************************/

  short int   minus1 = -1;            /* default null indic setting */

  char        *result;                /* result of this function     */
  short int   *niResult;              /* indic var, result           */
  char        *sqlstate;              /* SQLSTATE                    */
  char        fnName[138];            /* function name               */
  char        specificName[129];      /* specific name of function   */
  char        *message;               /* diagnostic message          */

  short int   status = OK;            /* DSN8DUTI run status         */


  /******************************************************************
  * Walk down the argv list, locating the input and output parms     *
  ******************************************************************/
  argc--;                            /* convert argc to base 0 index*/

  message = (char *)argv[argc--];    /* out: point to UDF diag msg  */

  strcpy( specificName,              /* in: save UDF specific name  */
          argv[argc--]);

  strcpy( fnName,argv[argc--] );     /* in: save UDF function name  */

  sqlstate = (char *)argv[argc--];   /* out: point to UDF sqlstate  */

  niResult                           /* out: point to null indicator*/
    = (short int *)argv[argc--];     /*      variable for result    */

  if( argc == 7 )                    /* if 3 input parms passed     */
    niObjLocation                    /* ..in: point to null indic.  */
      = (short int *)argv[argc--];   /*       var for object loc'n  */
  else                               /* otherwise it wasn't passed  */
    niObjLocation = &minus1;         /* ..so define it as null      */

  if( argc >= 5 )                    /* if 2 or 3 input parms passed*/
    niObjSchema                      /* ..in: point to null indic.  */
      = (short int *)argv[argc--];   /*       var for object schema */
  else                               /* otherwise it wasn't passed  */
    niObjSchema = &minus1;           /* ..so define it as null      */

  niObjName                          /* in: point to null indicator */
    = (short int *)argv[argc--];     /*     var for object name     */

  result = (char *)argv[argc--];     /* out: point to UDF result    */

  if( argc == 3 )                    /* if 3 input parms passed     */
    strcpy( hvObjLocation,           /* ..in: save object location  */
            argv[argc--] );          /*       name                  */
```

```
else                               /* otherwise it wasn't passed  */
  hvObjLocation[0] = NULLCHAR;     /* ..so define it as null      */

if( argc >= 2 )                    /* if 2 or 3 input parms passed*/
  strcpy( hvObjSchema,             /* ..in: save object schema    */
          argv[argc--] );          /*                             */
else                               /* otherwise it wasn't passed  */
  hvObjSchema[0] = NULLCHAR;       /* ..so define it as null      */

strcpy( hvObjName,argv[argc] );    /* in: save object name        */


/*******************************************************************
* Initialize output parms                                          *
*******************************************************************/
message[0] = NULLCHAR;
strcpy( sqlstate,"00000" );
*niResult = 0;
result[0] = NULLCHAR;

/*******************************************************************
* If no object name provided, return null result                   *
*******************************************************************/
if( ( *niObjName != 0 ) || ( strlen( hvObjName ) == 0 ) )
  status = NOT_OK;
/*******************************************************************
* If no object schema provided, assign default schema              *
*******************************************************************/
if( ( *niObjSchema != 0 ) || ( strlen( hvObjSchema ) == 0 ) )
  EXEC SQL SET :hvObjSchema = CURRENT SQLID;
/*******************************************************************
* Concatenate "wildcard" to object location                        *
*******************************************************************/
strcat( hvObjLocation,"%" );

/*******************************************************************
* Look for alias with the object name (and schema (and location))  *
*******************************************************************/
if( status == OK )
  {
    EXEC SQL SELECT  TBNAME,
                     TBCREATOR,
                     LOCATION
               INTO :hvTBNAME,
                    :hvTBCREATOR,
                    :hvLOCATION
               FROM  SYSIBM.SYSTABLES
              WHERE  NAME        = :hvObjName
                AND  CREATOR     = :hvObjSchema
                AND  LOCATION LIKE :hvObjLocation
                AND  TYPE        = 'A';

    if( SQLCODE == 0 )
      /*************************************************************
      * If such an alias was found ...                             *
      *************************************************************/
      if( strncmp( specificName,"DSN8DUTIN",9 ) == 0 )
        /***********************************************************
        * TABLE_NAME UDF: return true name of table or view        *
        ***********************************************************/
        strcpy( result,hvTBNAME );
      else if( strncmp( specificName,"DSN8DUTIS",9 ) == 0 )
        /***********************************************************
        * TABLE_SCHEMA UDF: return true schema of table or view    *
        ***********************************************************/
        strcpy( result,hvTBCREATOR );
      else if( strncmp( specificName,"DSN8DUTIL",9 ) == 0 )
        /***********************************************************
        * TABLE_LOCATION UDF: return true loc'n of table or view   *
        ***********************************************************/
        strcpy( result,hvLOCATION );
      else
        /***********************************************************
        * Unknown UDF: signal error                                *
        ***********************************************************/
        {
          status = NOT_OK;
          strcpy( sqlstate,"38601" );
          sprintf( message,
                   "DSN8DUTI Error: Invocation by unexpected UDF ",
                                "having specific name %s",
                   specificName );
```

```
            }

        else if( SQLCODE == 100 )
          /*************************************************************
          * If no such alias was found ...                             *
          *************************************************************/
          if( strncmp( specificName,"DSN8DUTIN",9 ) == 0 )
            /***********************************************************
            * TABLE_NAME UDF: return starting point                    *
            ***********************************************************/
            strcpy( result,hvObjName );
          else if( strncmp( specificName,"DSN8DUTIS",9 ) == 0 )
            /***********************************************************
            * TABLE_SCHEMA UDF: return schema of starting point        *
            ***********************************************************/
            strcpy( result,hvObjSchema );
          else if( strncmp( specificName,"DSN8DUTIL",9 ) == 0 )
            /***********************************************************
            * TABLE_LOCATION UDF: Remove trailing search wildcard byte *
            *                 and return location of starting point    *
            ***********************************************************/
            {
              hvObjLocation[strlen(hvObjLocation)-1] = NULLCHAR;
              strcpy( result,hvObjLocation );
            }
          else
            /***********************************************************
            * Unknown UDF: signal error                                *
            ***********************************************************/
            {
              status = NOT_OK;
              strcpy( sqlstate,"38601" );
              sprintf( message,
                       "DSN8DUTI Error: Invocation by unexpected UDF ",
                                       "having specific name %s",
                       specificName );
            }

        else
          /*************************************************************
          * If unexpected SQLCODE, issue message                       *
          *************************************************************/
          {
            status = NOT_OK;
            strcpy( sqlstate,"38602" );
            sprintf( message,
                     "DSN8DUTI Error: Unexpected SQLCODE, %d, "
                                     "from SQL SELECT",
                     SQLCODE );
          }
      } /* end if( status == OK ) */

    /*******************************************************************
    * If null starting point or unexpected SQLCODE, return null result *
    *******************************************************************/
    if( status == NOT_OK )
      {
        result[0] = NULLCHAR;
        *niResult = -1;
      }
    else
      {
        *niResult = 0;
        strcpy( sqlstate,"00000" );
      }

  } /* end DSN8DUTI */
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSN8DUWC

Invokes the sample UDF table function WEATHER to demon- strate how a UDF and UDF table handling using static SQL.

```
/*********************************************************************
* Module name = DSN8DUWC (DB2 sample program)                        *
*                                                                    *
* DESCRIPTIVE NAME = Client for sample UDF table function WEATHER    *
*                                                                    *
*                                                                    *
*    LICENSED MATERIALS - PROPERTY OF IBM                            *
*    5645-DB2                                                        *
*    (C) COPYRIGHT 1998 IBM CORP.  ALL RIGHTS RESERVED.              *
*                                                                    *
*    STATUS = VERSION 6                                              *
*                                                                    *
* Function: Invokes the sample UDF table function WEATHER to demon-  *
*           strate how a UDF and UDF table handling using static SQL.*
*                                                                    *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   *
* Module size: See linkedit output                                   *
*  Attributes: Re-entrant and re-usable                              *
*                                                                    *
* Entry Point: DSN8DUWC                                              *
*     Purpose: See Function                                          *
*     Linkage: DB2SQL                                                *
*              Invoked via SQL UDF call                              *
*                                                                    *
*                                                                    *
*  Parameters: DSN8DUWC uses the C "main" argument convention of     *
*              argv (argument vector) and argc (argument count).     *
*                                                                    *
*              - ARGV[0]  = (input) pointer to a char[9], null-      *
*                           terminated string having the name of     *
*                           this program (DSN8DUWC)                  *
*              - ARGV[1]  = (input) pointer to a char[45], null-     *
*                           terminated string having the name of the *
*                           source data for the weather reports.     *
*                                                                    *
* Normal Exit: Return Code: 0000                                     *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code: 0008                                     *
*              - Message: DSN8DUWC failed: Invalid parameter count   *
*                                                                    *
*              - Message: <formatted SQL text from DSNTIAR>          *
*                                                                    *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: DSNTIAR: DB2 msg text formatter   *
*             - Data areas       : None                              *
*             - Control blocks   : None                              *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8DUWC:                                                        *
*   - Verify that 2 input parameters (program name and weather data  *
*     set name) were passed.                                         *
*     - if not, issue diagnostic message and end with code 0008      *
*   - Open WEATHER_CURSOR, the client cursor for the WEATHER UDF     *
*     table function, passing the weather data set name as a host    *
*     variable                                                       *
*     - if unsuccessful, call sql_error to issue a diagnostic mes-   *
*       sage, then end with code 0008.                               *
*   - Do while not end of cursor                                     *
*     - Read the cursor                                              *
*     - If successful, print the result                              *
*     - else if not end of cursor, call sql_error to issue a diag-   *
*       nostic message, then end with code 0008.                     *
*   - Close the cursor                                               *
*     - if unsuccessful, call sql_error to issue a diagnostic mes-   *
```

```
*       sage, then end with code 0008.                              *
*   End DSN8DUWC                                                    *
*                                                                   *
*   sql_error:                                                      *
*   - call DSNTIAR to format the unexpected SQLCODE.                *
*   End sql_error                                                   *
*                                                                   *
*********************************************************************/
/********************** C library definitions ***********************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***************************** Equates ******************************/
#define         NULLCHAR     '\0'     /* Null character             */

#define         OUTLEN        80      /* Length of output line      */
#define         DATA_DIM      10      /* Number of message lines    */

#define         NOT_OK         0      /* Run status indicator: Error*/
#define         OK             1      /* Run status indicator: Good */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/************************ DB2 Host Variables ************************/
EXEC SQL BEGIN DECLARE SECTION;
  char          hvWeatherDSN[44];     /* host var for weather dsn   */
  short int     niWeatherDSN    = 0;  /* indic var for weather dsn  */

  char          hvCity[31];           /* host var for name of city  */
  short int     niCity          = 0;  /* indic var for city name    */

  long int      hvTemp_in_f     = 0;  /* host var, fahrenheit temp  */
  short int     niTemp_in_f     = 0;  /* indic var for temperature  */

  long int      hvHumidity      = 0;  /* host var, percent humidity */
  short int     niHumidity      = 0;  /* indic var for humidity     */

  char          hvWind[5];            /* host var, wind direction   */
  short int     niWind          = 0;  /* indic var for wind direct  */

  long int      hvWind_velocity = 0;  /* host var, wind velocity    */
  short int     niWind_velocity = 0;  /* indic var for wind velocity*/

  double        hvBarometer     = 0;  /* host var, barometric press */
  short int     niBarometer     = 0;  /* indic var for baro pressure*/

  char          hvForecast[26];       /* host var, forecast         */
  short int     niForecast      = 0;  /* indic var for forecast     */

EXEC SQL END DECLARE SECTION;


/******************* DB2 SQL Cursor Declarations ********************/

EXEC SQL BEGIN DECLARE SECTION;

EXEC SQL DECLARE  WEATHER_CURSOR
          CURSOR  FOR
          SELECT  CITY,                                                 00008900
                  TEMP_IN_F,                                            00008900
                  HUMIDITY,                                             00008900
                  WIND,                                                 00008900
                  WIND_VELOCITY,                                        00008900
                  BAROMETER,
                  FORECAST
            FROM  TABLE( DSN8.WEATHER(:hvWeatherDSN) ) AS W
           WHERE  CITY = 'Juneau, AK';

EXEC SQL END DECLARE SECTION;


/********************** DB2 Message Formatter ***********************/
struct          error_struct          /* DSNTIAR message structure  */
{
  short int     error_len;
  char          error_text[DATA_DIM][OUTLEN];
}               error_message = {DATA_DIM * (OUTLEN)};
```

```
#pragma         linkage( dsntiar, OS )

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );


/******************** DSN8DUWC Global Variables *********************/
short int       status = OK;          /* DSN8DUWC run status        */

long int        completion_code = 0;  /* DSN8DUWC return code       */


/******************* DSN8DUWC Function Prototypes *******************/
int main( int argc, char *argv[] );
void sql_error( char locmsg[] );


int main( int argc, char *argv[] )
/*********************************************************************
*                                                                    *
*********************************************************************/
{
  if( argc == 2 )
    strcpy( hvWeatherDSN,argv[1] );
  else
    {
      printf( "DSN8DUWC failed: Invalid parameter count\n" );
      status = NOT_OK;
    }

  if( status == OK )
    {
      EXEC SQL OPEN WEATHER_CURSOR;
      if( SQLCODE != 0 )
        sql_error( " *** Open weather cursor" );
    }

  while( SQLCODE == 0  &&  status == OK )
    {
      EXEC SQL FETCH  WEATHER_CURSOR
                INTO :hvCity         :niCity,
                     :hvTemp_in_f    :niTemp_in_f,
                     :hvHumidity     :niHumidity,
                     :hvWind         :niWind,
                     :hvWind_velocity:niWind_velocity,
                     :hvBarometer    :niBarometer,
                     :hvForecast     :niForecast;
      if( SQLCODE == 0 )
        {
          printf( "Weather Report for %s\n", hvCity );
          printf( "... Temperature   : %d\n", hvTemp_in_f );
          printf( "... Humidity      : %d\n", hvHumidity );
          printf( "... Wind direction: %s\n", hvWind );
          printf( "... Wind velocity : %d\n", hvWind_velocity );
          printf( "... Barometer     : %.2f\n", hvBarometer );
          printf( "... Forecast      : %s\n", hvForecast );
        }
      else if( SQLCODE != 100 )
        sql_error( " *** Fetch from weather cursor" );
    }

  if( status == OK )
    {
      EXEC SQL CLOSE WEATHER_CURSOR;
      if( SQLCODE != 0 )
        sql_error( " *** Close weather cursor" );
    }

  if( status != OK )
    completion_code = 8;

  return( completion_code );

} /* end main */


/*********************************************************************
**********************************************************************
** SQL error handler                                                **
**********************************************************************
*********************************************************************/
```

```
void sql_error( char locmsg[] )                                /*proc*/
{


  short int   rc;                    /* DSNTIAR Return code        */
  int         j,k;                   /* Loop control               */
  static int  lrecl = OUTLEN;        /* Width of message lines     */

  /*****************************************************************
  * set status to prevent further processing                       *
  ******************************************************************/
  status = NOT_OK;

  /*****************************************************************
  * print the locator message                                      *
  ******************************************************************/
  printf( " %.80s\n", locmsg );

  /*****************************************************************
  * format and print the SQL message                               *
  ******************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<DATA_DIM; j++ )
      {
        for( k=0; k<OUTLEN; k++ )
          putchar(error_message.error_text[j][k] );
        putchar('\n');
      }
  else
    {
      printf( " *** ERROR: DSNTIAR could not format the message\n" );
      printf( " ***        SQLCODE is %d\n",SQLCODE );
      printf( " ***        SQLERRM is \n" );
      for( j=0; j<sqlca.sqlerrml; j++ )
        printf( "%c", sqlca.sqlerrmc[j] );
      printf( "\n" );
    }

} /* end of sql_error */
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DUWF

Returns weather information for various cities, as read from the data set passed as the argument to input para- meter 'weatherDSN'.

```
/********************************************************************* 00000100
* Module name = DSN8DUWF (DB2 sample program)                        * 00000200
*                                                                    * 00000300
* DESCRIPTIVE NAME = Weather (DB2 user-defined table function)       * 00000400
*                                                                    * 00000500
*     5675-DB2                                                       * 00000600
*     (C) COPYRIGHT 1998, 2000 IBM CORP.                             * 00000700
*                                                                    * 00000800
*     STATUS = VERSION 7                                             * 00000900
*                                                                    * 00001000
* Function: Returns weather information for various cities, as read  * 00001100
*           from the data set passed as the argument to input para-  * 00001200
*           meter 'weatherDSN'.  The data includes the name of a     * 00001300
*           city followed by its weather information: Temperature in * 00001400
*           fahrenheit, percent humidity, wind direction, wind velo- * 00001500
*           city, barometric pressure, and the forecast.  See the    * 00001600
*           structure 'weatherRec' for the record format.            * 00001700
*                                                                    * 00001800
*           File pointer information is retained between calls in   * 00001900
*           the UDF scratchpad area.                                 * 00002000
*                                                                    * 00002100
*           Data read from the input data set is returned by this    * 00002200
*           function as a DB2 table with the following structure:    * 00002300
*           with this structure:                                     * 00002400
*                                                                    * 00002500
*           CITY          VARCHAR(30),                               * 00002600
*           TEMP_IN_F     INTEGER,                                   * 00002700
*           HUMIDITY      INTEGER,                                   * 00002800
```

```
 *           WIND          VARCHAR(5),                                 * 00002900
 *           WIND_VELOCITY INTEGER,                                    * 00003000
 *           BAROMETER     FLOAT,                                      * 00003100
 *           FORECAST      VARCHAR(25)                                 * 00003200
 *                                                                     * 00003300
 *           Clients invoking this function can use standard SQL      * 00003400
 *           syntax to create a desired result set.                    * 00003500
 *                                                                     * 00003600
 *           Example invocation:                                       * 00003700
 *                                                                     * 00003800
 *           EXEC SQL DECLARE  WEATHER_CURSOR                          * 00003900
 *                     CURSOR  FOR                                     * 00004000
 *                     SELECT  CITY,                                   * 00004100
 *                             FORECAST                                * 00004200
 *                       FROM  TABLE( DSN8.WEATHER(:hvWeatherDSN) )   * 00004300
 *                         AS  W                                       * 00004400
 *                      WHERE  CITY = 'Juneau, AK';                    * 00004500
 *                                                                     * 00004600
 * Notes:                                                              * 00004700
 *   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       * 00004800
 *                                                                     * 00004900
 *   Restrictions:                                                     * 00005000
 *                                                                     * 00005100
 * Module type: C program                                              * 00005200
 *   Processor: IBM C/C++ for OS/390 V1R3 or higher                    * 00005300
 * Module size: See linkedit output                                    * 00005400
 *  Attributes: Re-entrant and re-usable                               * 00005500
 *                                                                     * 00005600
 * Entry Point: DSN8DUWF                                               * 00005700
 *     Purpose: See Function                                           * 00005800
 *     Linkage: DB2SQL                                                 * 00005900
 *              Invoked via SQL UDF call                               * 00006000
 *                                                                     * 00006100
 *                                                                     * 00006200
 *       Input: Parameters explicitly passed to this function:        * 00006300
 *              - *weatherDSN  : pointer to a char[45], null-termi-   * 00006400
 *                               nated string having the name of the  * 00006500
 *                               source data for the weather reports. * 00006600
 *              - *niWeatherDSN: pointer to a short integer having    * 00006700
 *                               the null indicator variable for      * 00006800
 *                               *weatherDSN.                          * 00006900
 *              - *fnName      : pointer to a char[138], null-termi-  * 00007000
 *                               nated string having the UDF family   * 00007100
 *                               name of this function.                * 00007200
 *              - *specificName: pointer to a char[129], null-termi-  * 00007300
 *                               nated string having the UDF specific * 00007400
 *                               name of this function.                * 00007500
 *                                                                     * 00007600
 *                                                                     * 00007700
 *      Output: Parameters explicitly passed by this function:        * 00007800
 *              - *city        : pointer to a char[31], null-termi-   * 00007900
 *                               nated string to receive the name of  * 00008000
 *                               the city.                             * 00008100
 *              - *temp_in_f   : pointer to a long integer to receive * 00008200
 *                               the temperature in fahrenheit for    * 00008300
 *                               the city.                             * 00008400
 *              - *humidity    : pointer to a long integer to receive * 00008500
 *                               the percent humidity for the city.   * 00008600
 *              - *wind        : pointer to a char[6], null-termi-    * 00008700
 *                               nated string to receive the wind di- * 00008800
 *                               rection for the city.                 * 00008900
 *              - *wind_velocity: pointer to a long integer to receive* 00009000
 *                               the wind velocity for city.          * 00009100
 *              - *barometer   : pointer to a double word to receive  * 00009200
 *                               the barometric pressure for the city.* 00009300
 *              - *forecast    : pointer to a char[26], null-termi-   * 00009400
 *                               nated string to receive the forecast * 00009500
 *                               for the city.                         * 00009600
 *              - *niCity      : pointer to a short integer to re-    * 00009700
 *                               ceive the null indicator variable    * 00009800
 *                               for *city.                            * 00009900
 *              - *niTemp_in_f : pointer to a short integer to re-    * 00010000
 *                               ceive the null indicator variable    * 00010100
 *                               for *temp_in_f.                       * 00010200
 *              - *niHumidity  : pointer to a short integer to re-    * 00010300
 *                               ceive the null indicator variable    * 00010400
 *                               for *humidity.                        * 00010500
 *              - *niWind      : pointer to a short integer to re-    * 00010600
 *                               ceive the null indicator variable    * 00010700
 *                               for *wind.                            * 00010800
 *              - *niWind_velocity: pointer to a short integer to re- * 00010900
 *                               ceive the null indicator variable    * 00011000
```

```
*                             for *wind_velocity.                  *
*             - *niBarometer : pointer to a short integer to re-    *
*                             ceive the null indicator variable    *
*                             for *barometer.                      *
*             - *niForecast  : pointer to a short integer to re-    *
*                             ceive the null indicator variable    *
*                             for *forecast.                       *
*             - *sqlstate    : pointer to a char[06], null-termi-   *
*                             nated string to receive the SQLSTATE.*
*             - *message     : pointer to a char[70], null-termi-   *
*                             nated string to receive a diagnostic *
*                             message if one is generated by this  *
*                             function.                            *
*                                                                  *
* Normal Exit: Return Code: SQLSTATE = 00000                       *
*             - Message: none                                      *
*                                                                  *
*             Return Code: SQLSTATE = 02000 (end of input)         *
*             - Message: none                                      *
*                                                                  *
*  Error Exit: Return Code: SQLSTATE = 38601                       *
*             - Message: DSN8DUWF Error: Unable to allocate DD      *
*                                   <ddname>: Error code=<x>,      *
*                                   info code=<y>                  *
*                                                                  *
*             Return Code: SQLSTATE = 38602                        *
*             - Message: DSN8DUWF Error: Error opening weather data *
*                                   set                            *
*                                                                  *
*             Return Code: SQLSTATE = 38603                        *
*             - Message: DSN8DUWF Error: Error reading weather data *
*                                   set                            *
*                                                                  *
*             Return Code: SQLSTATE = 38604                        *
*             - Message: DSN8DUWF Error: Error closing weather data *
*                                   set                            *
*                                                                  *
*             Return Code: SQLSTATE = 38605                        *
*             - Message: DSN8DUWF Error: FREE failed for DDNAME <x>.*
*                                   Error code=<y>, info code=     *
*                                   <z>                            *
*                                                                  *
*    External References:                                          *
*            - Routines/Services: dyninit: IBM C/C++, dynit.h       *
*                                - Initializes control block for   *
*                                  dynamic file allocation         *
*                                dynalloc: IBM C/C++, dynit.h      *
*                                - Dynamic file allocation         *
*                                dynfree: IBM C/C++, dynit.h       *
*                                - Dynamic file deallocation       *
*            - Data areas       : None                              *
*            - Control blocks   : __dyn_t: IBM C/C++, dynint.h      *
*                                - for dynamic file allocation     *
*                                                                  *
*  Pseudocode:                                                     *
*   DSN8DUWF:                                                      *
*   - If SQLUDF call type is SQLUDF_TF_FIRST (-2)                  *
*     - call allocWeatherDSN to allocate the data set name passed as *
*       the argument to the weatherDSN parameter                   *
*   - Else if SQLUDF call type is SQLUDF_TF_OPEN (-1)              *
*     - call openWeatherDS to open the weather data set            *
*   - Else if SQLUDF call type is SQLUDF_TF_FETCH (0)              *
*     - call readWeatherDS to read the next record from the weather *
*       data set                                                   *
*     - if EOF, set sqlstate to 02000 to signal end of cursor to   *
*       client                                                     *
*     - else call buildReturnRow to populate the UDF table function *
*       output parameters with data from the input record          *
*   - Else if SQLUDF call type is SQLUDF_TF_CLOSE (1)              *
*     - call closeWeatherDS to close the weather data set          *
*   - Else (SQLUDF call type is SQLUDF_TF_FINAL (2) )              *
*     - call freeWeatherDS to deallocate the weather data set      *
*   End DSN8DUWF                                                   *
*                                                                  *
*   allocWeatherDS:                                                *
*   - if the data set name passed in as the argument to the input  *
*     parameter weatherDSN is for a partitioned data set, extract  *
*     the member name                                              *
*   - use dynalloc to dynamically allocate the data set (and member, *
*     if applicable)                                               *
*   - if allocation error occurs, issue sqlstate 38601 and a diag- *
*     nostic message                                               *
```

```
*   End allocWeatherDS                                                *
*                                                                     *
*   openWeatherDS:                                                    *
*   - open the weather data set                                       *
*   - if the data cannot be opened, issue sqlstate 38602 and a diag- *
*     nostic message                                                  *
*   End openWeatherDS                                                 *
*                                                                     *
*   readWeatherDS:                                                    *
*   - read the next record from the data set                          *
*     - this implicitly updates the file pointer variable, which is  *
*       maintained in the UDF scratchpad area                         *
*   - if the data set cannot be read, issue sqlstate 38603 and a     *
*     diagnostic message                                              *
*   End readWeatherDS                                                 *
*                                                                     *
*   buildReturnRow:                                                   *
*   - extract weather data fields from the weather data set          *
*   - perform appropriate data type conversions                       *
*   - copy the (converted) data to the appropriate output parameters *
*   End buildReturnRow                                                *
*                                                                     *
*   closeWeatherDS:                                                   *
*   - close the weather data set                                      *
*   - if the data cannot be closed, issue sqlstate 38604 and a diag- *
*     nostic message                                                  *
*   End closeWeatherDS                                                *
*                                                                     *
*   freeWeatherDS:                                                    *
*   - use dynfree to dynamically deallocate the weather data set     *
*   - if deallocation error occurs, issue sqlstate 38605 and a diag- *
*     nostic message                                                  *
*   End freeWeatherDS                                                 *
*                                                                     *
*********************************************************************/

#pragma linkage(DSN8DUWF,fetchable)

/********************** C library definitions ***********************/
#include <dynit.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/***************************** Equates *****************************/
#define         NO              0       /* Negative                  */
#define         YES             1       /* Affirmative               */

#define         NULLCHAR      '\0'      /* Null character            */

#define         SQLUDF_TF_FIRST -2      /* First call                */
#define         SQLUDF_TF_OPEN  -1      /* Open table call           */
#define         SQLUDF_TF_FETCH  0      /* Fetch next row call       */
#define         SQLUDF_TF_CLOSE  1      /* Close table call          */
#define         SQLUDF_TF_FINAL  2      /* Final call                */

/************************ Weather Data Set *************************/
struct scr                              /* Struct for scratchpad area */
{
  long          len;                    /* Length of scratchpad data  */
  FILE          *WEATHRin;              /* ptr to weather data set    */
  char          not_used[100];          /* filler                     */
};

char            WEATHRinBuffer[8188]; /* Input buffer for weather ds*/
short int       moreWeatherRecs = YES;/* EOF indicator, weather ds  */

typedef struct                          /* Weather record structure   */
{
  char          cityField[30];          /* name of city        01-30 */
  char          filler1[1];             /*                     31-31 */
  char          temp_in_fField[3];      /* temp in fahrenheit  32-34 */
  char          filler2[1];             /*                     35-35 */
  char          humidityField[3];       /* percent humidity    36-38 */
  char          filler3[1];             /*                     39-39 */
  char          windField[5];           /* wind direction      40-44 */
  char          filler4[1];             /*                     45-45 */
  char          windVelocityField[3];   /* wind velocity       46-48 */
  char          filler5[1];             /*                     49-49 */
  char          barometerField[5];      /* baromtric pressure  50-54 */
  char          filler6[1];             /*                     55-55 */
  char          forecastField[25];      /* forecast            56-80 */
```

```
} weatherRec;

weatherRec    *pweatherRec = (weatherRec *)&WEATHRinBuffer;


/*************************** Functions *****************************/
void *DSN8DUWF                          /* Weather function          */
( char          *weatherDSN,            /* in: input ds, weather data */
  char          *city,                  /* out: name of city          */
  long int      *temp_in_f,             /* out: temp in fahrenheit    */
  long int      *humidity,              /* out: relative humidity     */
  char          *wind,                  /* out: wind direction        */
  long int      *wind_velocity,         /* out: wind velocity         */
  double        *barometer,             /* out: barometric pressure   */
  char          *forecast,              /* out: forecast              */
  short int     *niWeatherDSN,          /* in: indic var, weather dsn */
  short int     *niCity,                /* out: indic var, city name  */
  short int     *niTemp_in_f,           /* out: indic var, temperature*/
  short int     *niHumidity,            /* out: indic var, humidity   */
  short int     *niWind,                /* out: indic var, wind dir   */
  short int     *niWind_velocity,       /* out: indic var, wind veloc */
  short int     *niBarometer,           /* out: indic var, baro press */
  short int     *niForecast,            /* out: indic var, forecast   */
  char          *sqlstate,              /* out: SQLSTATE              */
  char          *fnName,                /* in: family name of function*/
  char          *specificName,          /* in: specific name of func  */
  char          *msgtext,               /* out: diagnostic message    */
  struct scr    *scratchptr,            /* i/o: scratchpad area       */
  long          *callType               /* i/o: call type parameter   */
);

void allocWeatherDS                     /* Dynam allocates weather ds */
( char          *weatherDSN,            /* in: name of weather ds     */
  char          *sqlstate,              /* out: sqlstate              */
  char          *msgtext                /* out: diag message text     */
);

void openWeatherDS                      /* Opens weather data set     */
( struct scr    *scratchptr,            /* in: ptr to scratch pad     */
  char          *sqlstate,              /* out: sqlstate              */
  char          *msgtext                /* out: diag message text     */
);

void readWeatherDS                      /* Reads from weather data set*/
( struct scr    *scratchptr,            /* in: ptr to scratch pad     */
  char          *sqlstate,              /* out: sqlstate              */
  char          *msgtext                /* out: diag message text     */
);

void buildReturnRow                     /* Builds function return row */
( char          *city,                  /* out: name of city          */
  long int      *temp_in_f,             /* out: temp in fahrenheit    */
  long int      *humidity,              /* out: relative humidity     */
  char          *wind,                  /* out: wind direction        */
  long int      *wind_velocity,         /* out: wind velocity         */
  double        *barometer,             /* out: barometric pressure   */
  char          *forecast,              /* out: forecast              */
  short int     *niCity,                /* out: indic var, city name  */
  short int     *niTemp_in_f,           /* out: indic var, temperature*/
  short int     *niHumidity,            /* out: indic var, humidity   */
  short int     *niWind,                /* out: indic var, wind dir   */
  short int     *niWind_velocity,       /* out: indic var, wind veloc */
  short int     *niBarometer,           /* out: indic var, baro press */
  short int     *niForecast             /* out: indic var, forecast   */
);

void closeWeatherDS                     /* Closes weather data set    */
( struct scr    *scratchptr,            /* in: ptr to scratch pad     */
  char          *sqlstate,              /* out: sqlstate              */
  char          *msgtext                /* out: diag message text     */
);

void freeWeatherDS                      /* Dynam frees the weather ds */
( char          *sqlstate,              /* out: sqlstate              */
  char          *msgtext                /* out: diag message text     */
);


void *DSN8DUWF
( char          *weatherDSN,            /* in: input ds, weather data */
  char          *city,                  /* out: name of city          */
  long int      *temp_in_f,             /* out: temp in fahrenheit    */
```

```
  long int      *humidity,           /* out: relative humidity     */
  char          *wind,               /* out: wind direction        */
  long int      *wind_velocity,      /* out: wind velocity         */
  double        *barometer,          /* out: barometric pressure   */
  char          *forecast,           /* out: forecast              */
  short int     *niWeatherDSN,       /* in: indic var, weather dsn */
  short int     *niCity,             /* out: indic var, city name  */
  short int     *niTemp_in_f,        /* out: indic var, temperature*/
  short int     *niHumidity,         /* out: indic var, humidity   */
  short int     *niWind,             /* out: indic var, wind dir   */
  short int     *niWind_velocity,    /* out: indic var, wind veloc */
  short int     *niBarometer,        /* out: indic var, baro press */
  short int     *niForecast,         /* out: indic var, forecast   */
  char          *sqlstate,           /* out: SQLSTATE              */
  char          *fnName,             /* in: family name of function*/
  char          *specificName,       /* in: specific name of func  */
  char          *msgtext,            /* out: diagnostic message    */
  struct scr    *scratchptr,         /* i/o: scratchpad area       */
  long          *callType            /* i/o: call type parameter   */
)
/*******************************************************************
* Main routine for Weather table function                          *
*******************************************************************/
{
  /*****************************************************************
  * First call: Dynamically allocate the weather data set          *
  *****************************************************************/
  if( *callType == SQLUDF_TF_FIRST )
    {
      strcpy( sqlstate,"00000" );     /* Init sqlstate return var   */
      *msgtext  = NULLCHAR;           /* Init message text rtrn var */
      allocWeatherDS( weatherDSN, sqlstate, msgtext );
    }
  /*****************************************************************
  * Second call: Open the weather data set                         *
  *****************************************************************/
  else if( *callType == SQLUDF_TF_OPEN )
    {
      strcpy( sqlstate,"00000" );     /* Init sqlstate return var   */
      *msgtext  = NULLCHAR;           /* Init message text rtrn var */
      moreWeatherRecs = YES;          /* EOF indicator, weather ds  */
      openWeatherDS( scratchptr, sqlstate, msgtext );
    }
  /*****************************************************************
  * Subsequent calls: Read a record from the weather data set      *
  *****************************************************************/
  else if( *callType == SQLUDF_TF_FETCH )
    {
      readWeatherDS( scratchptr, sqlstate, msgtext );
      if( moreWeatherRecs == NO )     /* If no more weather data    */
        strcpy( sqlstate,"02000" );   /* ..signal for FINAL CALL    */
      else
        {
          buildReturnRow( city,
                          temp_in_f,
                          humidity,
                          wind,
                          wind_velocity,
                          barometer,
                          forecast,
                          niCity,
                          niTemp_in_f,
                          niHumidity,
                          niWind,
                          niWind_velocity,
                          niBarometer,
                          niForecast );
        }
    }
  /*****************************************************************
  * End of file: Close weather data set                            *
  *****************************************************************/
  else if( *callType == SQLUDF_TF_CLOSE )
    {
      closeWeatherDS( scratchptr, sqlstate, msgtext );
    }
  /*****************************************************************
  * Final call: De-allocate weather data set                       *
  *****************************************************************/
  else /* *callType == SQLUDF_TF_FINAL */
    {
      freeWeatherDS( sqlstate, msgtext );
```

```
    }
  return;
}


void allocWeatherDS
( char          *weatherDSN,          /* in: name of weather ds     */
  char          *sqlstate,            /* out: sqlstate              */
  char          *msgtext              /* out: diag message text     */
)
/*******************************************************************
* Dynamically allocates weatherDSN to the WEATHRIN DD.  If the value *
* in weatherDSN contains parentheses, it is assumed to specify a     *
* partitioned data set; otherwise it is assumed to specify a         *
* physical sequential data set.                                      *
*******************************************************************/
{
  __dyn_t       ip;                   /* pointer to control block   */
  char          DSname[45];           /* recv's copy of weather dsn */
  char          *tokPtr;              /* string ptr for token parser*/

  dyninit( &ip );                     /* Initialize control block   */
  ip.__ddname = "WEATHRIN";           /* Specify DDNAME of WEATHRIN */
  /*****************************************************************
  * Use the strtok func to separate the PDS member name, if any,     *
  * from the data set name                                           *
  *****************************************************************/
  strcpy( DSname,weatherDSN );        /* Get workcopy of weather dsn*/
  tokPtr = strtok( DSname,"(" );      /* Parse for open parenthesis */
  if( tokPtr == NULL )                /* If none found then         */
    ip.__dsname = DSname;             /* ...data set is not a PDS   */
  else                                /* Otherwise                  */
    {
      ip.__dsname = tokPtr;           /* ...token is name of a PDS  */
      tokPtr = strtok( NULL,")" );    /* ...parse for close paren   */
      ip.__member = tokPtr;           /* ...token is name of member */
    }
  ip.__status = __DISP_SHR;           /* Specify DISP=SHR           */

  /*****************************************************************
  * If dynamic allocation failed, generate an error message and quit *
  *****************************************************************/
  if( dynalloc(&ip) != 0 )
    {
      sprintf( msgtext,"Unable to allocate DD %s: "
                       "Error code=%hX, info code=%hX\n",
               ip.__ddname,
               ip.__errcode,
               ip.__infocode );
      strcpy( sqlstate,"38601" );
    }
} /* end allocWeatherDS */


void openWeatherDS
( struct scr    *scratchptr,          /* in: ptr to scratch pad     */
  char          *sqlstate,            /* out: sqlstate              */
  char          *msgtext              /* out: diag message text     */
)
/*******************************************************************
* Opens the weather data set, which has been allocated to the DD     *
* WEATHRIN, for record-type input, and assigns the file pointer to   *
* the scratchpad area indicated by scratchptr.                       *
*******************************************************************/
{
  scratchptr->WEATHRin = fopen("DD:WEATHRIN",
                               "rb,recfm=vb,lrecl=8188,type=record");

  if( scratchptr->WEATHRin == NULL )  /* If unable to open data set */
    {                                 /* ..set return msg and state */
      strcpy( msgtext,"Error opening weather data set" );
      strcpy( sqlstate,"38602" );
    }
} /* end openWeatherDS */


void readWeatherDS
( struct scr    *scratchptr,          /* in: ptr to scratch pad     */
  char          *sqlstate,            /* out: sqlstate              */
  char          *msgtext              /* out: diag message text     */
)
/*******************************************************************
```

```
* Reads the next record from the weather data set                   *
*********************************************************************/
{
  short int     recordLength = 0;     /* Receives len of current rec*/

  recordLength                      /*                           */
    = fread( WEATHRinBuffer,          /* Read into WEATHRinBuffer   */
             1,                     /* ..a record                */
             sizeof( WEATHRinBuffer ),/* ..<= len of WEATHRinBuffer */
             scratchptr->WEATHRin );  /* ..from the weather data set*/

  if( ferror(scratchptr->WEATHRin) )  /* If an error occurs         */
    {                                 /* ..set return msg and state */
      strcpy( msgtext,"Error reading weather data set" );
      strcpy( sqlstate,"38603" );
    }
  else if( feof(scratchptr->WEATHRin))/* Else if end of file reached*/
    moreWeatherRecs = NO;             /* ..get ready to quit        */
} /* end readWeatherDS */


void buildReturnRow                   /* Builds function return row */
( char          *city,                /* out: name of city          */
  long int      *temp_in_f,           /* out: temp in fahrenheit    */
  long int      *humidity,            /* out: relative humidity     */
  char          *wind,                /* out: wind direction        */
  long int      *wind_velocity,       /* out: wind velocity         */
  double        *barometer,           /* out: barometric pressure   */
  char          *forecast,            /* out: forecast              */
  short int     *niCity,              /* out: indic var, city name  */
  short int     *niTemp_in_f,         /* out: indic var, temperature*/
  short int     *niHumidity,          /* out: indic var, humidity   */
  short int     *niWind,              /* out: indic var, wind dir   */
  short int     *niWind_velocity,     /* out: indic var, wind veloc */
  short int     *niBarometer,         /* out: indic var, baro press */
  short int     *niForecast           /* out: indic var, forecast   */
)
/*********************************************************************
* Build a return row for the current call to the WEATHER table      *
* function.                                                         *
*********************************************************************/
{
  char          workBuff[6];          /* for datatype conversions   */

  /*******************************************************************
  * Move the city name to its table variable                        *
  *******************************************************************/
  strncpy( city,pweatherRec->cityField,30 );
  *niCity = 0;

  /*******************************************************************
  * Move the temperature to its table var after making it numeric    *
  *******************************************************************/
  memset( workBuff,'\0',6 );
  strncpy( workBuff,pweatherRec->temp_in_fField,3 );
  *temp_in_f = atoi( workBuff );
  *niTemp_in_f = 0;

  /*******************************************************************
  * Move the humidity factor to its table var after making it numeric*
  *******************************************************************/
  memset( workBuff,'\0',6 );
  strncpy( workBuff,pweatherRec->humidityField,3 );
  *humidity = atoi( workBuff );
  *niHumidity = 0;

  /*******************************************************************
  * Move the wind direction to its table variable                   *
  *******************************************************************/
  strncpy( wind,pweatherRec->windField,5 );
  *niWind = 0;

  /*******************************************************************
  * Move the wind velocity to its table var after making it numeric  *
  *******************************************************************/
  memset( workBuff,'\0',6 );
  strncpy( workBuff,pweatherRec->windVelocityField,3 );
  *wind_velocity = atoi( workBuff );
  *niWind_velocity = 0;

  /*******************************************************************
  * Move the forecast to its table variable                         *
```

```
  ******************************************************************/
  memset( workBuff,'\0',6 );
  strncpy( workBuff,pweatherRec->barometerField,5 );
  *barometer = atof( workBuff );
  *niBarometer = 0;

  /*****************************************************************
  * Move the forecast to its table variable                        *
  ******************************************************************/
  strncpy( forecast,pweatherRec->forecastField,25 );
  *niForecast = 0;

} /* end buildReturnRow */


void closeWeatherDS
( struct scr    *scratchptr,          /* in: ptr to scratch pad     */
  char          *sqlstate,            /* out: sqlstate              */
  char          *msgtext              /* out: diag message text     */
)
/*******************************************************************
* Closes the weather data set and resets the file pointer in the   *
* scratchpad area.                                                 *
*******************************************************************/
{
  if( fclose(scratchptr->WEATHRin) != 0 )
                                      /* If unable to close data set*/
    {                                 /* ..set return msg and state */
      strcpy( msgtext,"Error closing weather data set" );
      strcpy( sqlstate,"38604" );
    }
  else
    scratchptr->WEATHRin = NULLCHAR; /* Otherwise, reset file ptr   */
} /* end closeWeatherDS */


void freeWeatherDS
( char          *sqlstate,            /* out: sqlstate              */
  char          *msgtext              /* out: diag message text     */
)
/*******************************************************************
* Dynamically frees the weather data set, which has been allocated *
* to the WEATHRIN DD.                                              *
*******************************************************************/
{
  __dyn_t       free_ip;              /* pointer to control block   */

  dyninit( &free_ip );                /* Initialize control block   */
  free_ip.__ddname = "WEATHRIN";      /* Set DD name of weather ds  */

  if( dynfree(&free_ip) != 0 )
    {
      sprintf( msgtext,"FREE failed for DDNAME %s. "
                       "Error code=%hX, info code=%hX\n",
               free_ip.__errcode,
               free_ip.__errcode,
               free_ip.__infocode );
      strcpy( sqlstate,"38605" );
    }
} /* end freeWeatherDS */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EUDN

Returns the day of the week (Monday through Sunday) on which a given date in ISO format (YYYY-MM-DD) falls.

```
/*******************************************************************
* Module name = DSN8EUDN (DB2 sample program)                      *
*                                                                  *
* DESCRIPTIVE NAME = Query day of the week (UDF)                   *
*                                                                  *
*     LICENSED MATERIALS - PROPERTY OF IBM                         *
*     5675-DB2                                                     *
*     (C) COPYRIGHT 2000 IBM CORP.  ALL RIGHTS RESERVED.           *
```

```
*                                                                   *
*     STATUS = VERSION 7                                            *
*                                                                   *
* Function: Returns the day of the week (Monday through Sunday) on  *
*           which a given date in ISO format (YYYY-MM-DD) falls.    *
*                                                                   *
*           Example invocation:                                     *
*            EXEC SQL SET :dayname = DAYNAME( "2000-01-29" );       *
*            ==> dayname = Tuesday                                  *
* Notes:                                                            *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher      *
*                                                                   *
*   Restrictions: Assumes the Gregorian calendar was adopted in     *
*                 September, 1752.  Code modifications are required *
*                 to handle a different adoption date.              *
*                                                                   *
* Module type: C++ program                                          *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                  *
* Module size: See linkedit output                                  *
*  Attributes: Re-entrant and re-usable                             *
*                                                                   *
* Entry Point: DSN8EUDN                                             *
*     Purpose: See Function                                         *
*     Linkage: DB2SQL                                               *
*              Invoked via SQL UDF call                             *
*                                                                   *
*       Input: Parameters explicitly passed to this function:       *
*              - *ISOdateIn   : pointer to a char[11], null-termi-  *
*                               nated string having a date in ISO   *
*                               format.                             *
*              - *niISOdateIn : pointer to a short integer having   *
*                               the null indicator variable for     *
*                               *ISOdateIn.                         *
*              - *fnName      : pointer to a char[138], null-termi- *
*                               nated string having the UDF family  *
*                               name of this function.              *
*              - *specificName: pointer to a char[129], null-termi- *
*                               nated string having the UDF specific *
*                               name of this function.              *
*                                                                   *
*                                                                   *
*      Output: Parameters explicitly passed by this function:       *
*              - *dayNameOut  : pointer to a char[10], null-termi-  *
*                               nated string to receive the dayname *
*                               for ISOdateIn.                      *
*              - *niDayNameOut: pointer to a short integer to re-   *
*                               ceive the null indicator variable   *
*                               for *dayNameOut.                    *
*              - *sqlstate    : pointer to a char[06], null-termi-  *
*                               nated string to receive the SQLSTATE.*
*              - *message     : pointer to a char[70], null-termi-  *
*                               nated string to receive a diagnostic *
*                               message if one is generated by this *
*                               function.                           *
*                                                                   *
* Normal Exit: Return Code: SQLSTATE = 00000                        *
*              - Message: none                                      *
*                                                                   *
*  Error Exit: Return Code: SQLSTATE = 38601                        *
*              - Message: DSN8EUDN Error: No date entered           *
*              Return Code: SQLSTATE = 38602                        *
*              - Message: DSN8EUDN Error: Input date not valid      *
*                                        or not in ISO format"      *
*                                                                   *
*    External References:                                           *
*             - Routines/Services:                                  *
*               - strftime: Formatted time conversion routine       *
*                 - from IBM C/C++ for z/OS run-time library        *
*               - strptime: Date and time conversion routine        *
*                 - from IBM C/C++ for z/OS run-time library        *
*             - Data areas       : None                             *
*             - Control blocks   : None                             *
*                                                                   *
*                                                                   *
*  Pseudocode:                                                      *
*   DSN8EUDN:                                                       *
*   - Verify that a date was passed in:                             *
*     - if *ISOdateIn blank or niISOdateIn is not 0, no date passed: *
*       - issue SQLSTATE 38601 and a diagnostic message.            *
*   - Use strptime to validate the entry                            *
*     - if *ISOdateIn is not a valid ISO date:                      *
*       - issue SQLSTATE 38602 and a diagnostic message             *
```

```
*   - Parse out the year, month, and day                            *
*   - Compute the weekday number (0=Sunday, ..., 6=Saturday)        *
*   - Use strptime and strftime to convert the day number to the    *
*     full weekday name of the current locale.                      *
*   End DSN8EUDN                                                    *
*                                                                   *
*  Change log:                                                      *
*  2004-02-25: Rewritten due to demise of IBM Open Class library    *
*                                                                   *
********************************************************************/
extern "C" void DSN8EUDN             /* Establish linkage         */
( char       *ISOdateIn,             /* in: date to look up       */
  char       *dayNameOut,            /* out: ISOdateIn's day name  */
  short int  *niISOdateIn,           /* in: indic var, ISOdateIn   */
  short int  *niDayNameOut,          /* out: indic var, dayNameOut */
  char       *sqlstate,              /* out: SQLSTATE              */
  char       *fnName,                /* in: family name of function*/
  char       *specificName,          /* in: specific name of func  */
  char       *message                /* out: diagnostic message    */
);
#pragma linkage(DSN8EUDN,fetchable)  /* Establish linkage          */

/********************** C library definitions **********************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/****************************** Equates ******************************/
#define     NULLCHAR     '\0'         /* Null character             */

#define     MATCH          0          /* Comparison status: Equal   */
#define     NOT_OK         0          /* Run status indicator: Error*/
#define     OK             1          /* Run status indicator: Good */

/*********************** DSN8EUDN functions ************************/

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
void DSN8EUDN                        /* main routine               */
( char       *ISOdateIn,             /* in: date to look up        */
  char       *dayNameOut,            /* out: ISOdateIn's day name  */
  short int  *niISOdateIn,           /* in: indic var, ISOdateIn   */
  short int  *niDayNameOut,          /* out: indic var, dayNameOut */
  char       *sqlstate,              /* out: SQLSTATE              */
  char       *fnName,                /* in: family name of function*/
  char       *specificName,          /* in: specific name of func  */
  char       *message                /* out: diagnostic message    */
)
/********************************************************************
* Returns the weekday name of the date in ISOdateIn.                *
*                                                                   *
* Assumptions:                                                      *
* - *ISOdateIn       points to a char[11], null-terminated string   *
* - *dayNameOut      points to a char[10], null-terminated string   *
* - *niISOdateIn     points to a short integer                      *
* - *niDayNameOut    points to a short integer                      *
* - *sqlstate        points to a char[06], null-terminated string   *
* - *fnName          points to a char[138], null-terminated string  *
* - *specificName    points to a char[129], null-terminated string  *
* - *message         points to a char[70], null-terminated string   *
********************************************************************/
{

  /************************ local variables ************************/
  short int  status = OK;            /* DSN8EUDN run status        */
  struct tm  tmbuff;                 /* buffer for time.h tm struct*/
  char       *rc;                    /* gets strf/ptime return code*/

  char       *tokPtr;                /* string ptr for token parser*/
  char       workStr[11];            /* work copy of ISOdateIn parm*/

  int        yearInt;                /* numeric copy of 4-digit yr */
  char       yearStr[05];            /* string copy of 4-digit year*/

  int        monthInt;               /* numeric copy of month no.  */
  char       monthStr[03];           /* string copy of month no.   */

  int        dayInt;                 /* numeric copy of day no.    */
  char       dayStr[03];             /* string copy of day no.     */

  int        weekDayInt;             /* week day no (0=Sun...6=Sat)*/
```

```
char        weekDayStr[02];         /* string copy of week day no.*/

char        *isoFormat              /* format of isoDate:         */
            = "%Y-%m-%d";           /* %Y = YYYY, %m = MM, %d = DD*/
char        *weekDayFormat          /* format of weekday:         */
            = "%w";                 /* %w = weekday               */
char        *weekDayLongNameFormat  /* format of weekday long name*/
            = "%A";                 /* %A = weekday long name     */

/*******************************************************************
* Verify that something has been passed in                        *
*******************************************************************/
if( *niISOdateIn != 0 || ( strlen( ISOdateIn ) == 0 ) )
  {
    status = NOT_OK;
    strcpy( message,
            "DSN8EUDN Error: No date entered" );
    strcpy( sqlstate, "38601" );
  }

/*******************************************************************
* Verify that the input looks like a date                         *
*******************************************************************/
if( status == OK )
  { rc = strptime( ISOdateIn,isoFormat,&tmbuff );
    if( rc == NULL )                /* Unable to convert ISOdateIn*/
      {
        status = NOT_OK;
        strcpy( message,
                "DSN8EUDN Error: Input date not valid "
                                "or not in ISO format" );
        strcpy( sqlstate, "38602" );
      }
  }

/*******************************************************************
* Parse the 4-digit year, the month no., and day no. from ISOdateIn*
*******************************************************************/
if( status == OK )
  { strcpy( workStr,ISOdateIn );

    tokPtr  = strtok( workStr,"-" );
    strcpy( yearStr,tokPtr );
    yearInt  = atoi( yearStr );

    tokPtr  = strtok( NULL,"-" );
    strcpy( monthStr,tokPtr );
    monthInt = atoi( monthStr );

    tokPtr  = strtok( NULL,"-" );
    strcpy( dayStr,tokPtr );
    dayInt   = atoi( dayStr );
  }

/*******************************************************************
* Get the weekday name of ISOdateIn                               *
*******************************************************************/
if( status == OK )
  { /***************************************************************
    * Leap year allowance: Shift Jan and Feb to end of prev year   *
    ***************************************************************/
    if( monthInt < 3)
      { monthInt += 12;
        yearInt--;
      }

    /***************************************************************
    * Calculate weekday no. with Sunday basis                      *
    ***************************************************************/
    weekDayInt = (    ((13 * monthInt) + 3) / 5  /* xform months   */
                   + dayInt                      /* + days         */
                   + yearInt                     /* + years        */
                   + yearInt / 4                 /* + leapyear/4   */
                   - yearInt / 100               /* - leapyear/100 */
                   + yearInt / 400               /* + leapyear/400 */
                   + 1                           /* + Sunday basis */
                 ) % 7;                          /* % days per wk  */

    /***************************************************************
    * adjust for pre-gregorian calendar (September 1752)           *
    ***************************************************************/
    if( (yearInt < 1752) || (yearInt == 1752 && monthInt < 9) )
```

```
        { if( weekDayInt > 3 )
            weekDayInt = weekDayInt - 4;
          else
            weekDayInt = weekDayInt + 3;
        }

      /**************************************************************
      * convert day of week from numeric to string                  *
      **************************************************************/
      sprintf( weekDayStr,"%02d",weekDayInt );

      /**************************************************************
      * Convert day of week from numeric string to day name         *
      **************************************************************/
      rc = strptime( weekDayStr,weekDayFormat,&tmbuff );
      *rc = strftime( dayNameOut,10,weekDayLongNameFormat,&tmbuff );

    }

  /******************************************************************
  * If weekday name was obtained, clear the message buffer and sql-  *
  * state, and unset the SQL null indicator for dayNameOut.          *
  ******************************************************************/
  if( status == OK )
    {
      *niDayNameOut = 0;
      message[0] = NULLCHAR;
      strcpy( sqlstate,"00000" );
    }
  /******************************************************************
  * If errors occurred, clear the dayNameOut buffer and set the SQL  *
  * NULL indicator.  A diagnostic message and the SQLSTATE have been *
  * set where the error was detected.                                *
  ******************************************************************/
  else
    {
      dayNameOut[0] = NULLCHAR;
      *niDayNameOut = -1;
    }

  return;
}  /* end DSN8EUDN */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8EUMN

Returns the calendar name of the month name in which a given date in ISO format (YYYY-MM-DD) falls.

```
/*********************************************************************
* Module name = DSN8EUMN (DB2 sample program)                        *
*                                                                    *
* DESCRIPTIVE NAME = Query calendar month name (UDF)                 *
*                                                                    *
*                                                                    *
* LICENSED MATERIALS - PROPERTY OF IBM                               *
* 5675-DB2                                                           *
* (C) COPYRIGHT 1998, 2000 IBM CORP.  ALL RIGHTS RESERVED.           *
*                                                                    *
* STATUS = VERSION 7                                                 *
*                                                                    *
*                                                                    *
* Function: Returns the calendar name of the month name in           *
*           which a given date in ISO format (YYYY-MM-DD) falls.     *
*                                                                    *
*           Example invocation:                                      *
*            EXEC SQL SET :monthname = MONTHNAME( "2000-01-29" );    *
*            ==> monthname = January                                 *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C++ program                                           *
*   Processor: IBM C/C++ for OS/390 V1R3 or higher                   *
* Module size: See linkedit output                                   *
```

```
*  Attributes: Re-entrant and re-usable                             *
*                                                                   *
* Entry Point: DSN8EUMN                                             *
*     Purpose: See Function                                         *
*     Linkage: DB2SQL                                               *
*              Invoked via SQL UDF call                             *
*                                                                   *
*       Input: Parameters explicitly passed to this function:       *
*              - *ISOdateIn   : pointer to a char[11], null-termi-  *
*                               nated string having a date in ISO   *
*                               format.                             *
*              - *niISOdateIn : pointer to a short integer having   *
*                               the null indicator variable for     *
*                               *ISOdateIn.                         *
*              - *fnName      : pointer to a char[138], null-termi- *
*                               nated string having the UDF family  *
*                               name of this function.              *
*              - *specificName: pointer to a char[129], null-termi- *
*                               nated string having the UDF specific *
*                               name of this function.              *
*                                                                   *
*                                                                   *
*      Output: Parameters explicitly passed by this function:       *
*              - *monthNameOut: pointer to a char[10], null-termi-  *
*                               nated string to receive the month-  *
*                               name for ISOdateIn.                 *
*              - *niMonthNameOut: pointer to a short integer to re- *
*                               ceive the null indicator variable   *
*                               for *monthNameOut.                  *
*              - *sqlstate    : pointer to a char[06], null-termi-  *
*                               nated string to receive the SQLSTATE.*
*              - *message     : pointer to a char[70], null-termi-  *
*                               nated string to receive a diagnostic *
*                               message if one is generated by this *
*                               function.                           *
*                                                                   *
* Normal Exit: Return Code: SQLSTATE = 00000                        *
*              - Message: none                                      *
*                                                                   *
*  Error Exit: Return Code: SQLSTATE = 38601                        *
*              - Message: DSN8EUMN Error: No date entered           *
*              Return Code: SQLSTATE = 38602                        *
*              - Message: DSN8EUMN Error: Input date not valid      *
*                                        or not in ISO format"      *
*                                                                   *
*    External References:                                           *
*             - Routines/Services:                                  *
*               - strftime: Formatted time conversion routine       *
*                 - from IBM C/C++ for z/OS run-time library        *
*               - strptime: Date and time conversion routine        *
*                 - from IBM C/C++ for z/OS run-time library        *
*             - Data areas       : None                             *
*             - Control blocks   : None                             *
*                                                                   *
*                                                                   *
*  Pseudocode:                                                      *
*   DSN8EUMN:                                                       *
*   - Verify that a date was passed in:                             *
*     - if *ISOdateIn blank or niISOdateIn is not 0, no date passed: *
*       - issue SQLSTATE 38601 and a diagnostic message.            *
*   - Use strptime to validate the entry and convert the date to tm *
*     - if *ISOdateIn is not a valid ISO date:                      *
*       - issue SQLSTATE 38602 and a diagnostic message             *
*   - Use strftime to get the full monthname for the locale         *
*   End DSN8EUMN                                                    *
*                                                                   *
*  Change log:                                                      *
*  2004-02-25: Rewritten due to demise of IBM Open Class library    *
*                                                                   *
*********************************************************************/
extern "C" void DSN8EUMN             /* Establish linkage         */
( char        *ISOdateIn,            /* in: date to look up       */
  char        *monthNameOut,         /* out: ISOdateIn's month name*/
  short int   *niISOdateIn,          /* in: indic var, ISOdateIn  */
  short int   *niMonthNameOut,       /* out: indic var,monthNameOut*/
  char        *sqlstate,             /* out: SQLSTATE             */
  char        *fnName,               /* in: family name of function*/
  char        *specificName,         /* in: specific name of func */
  char        *message               /* out: diagnostic message   */
);

#pragma linkage(DSN8EUMN,fetchable)  /* Establish linkage         */
```

```
/********************** C library definitions ***********************/
#include <stdio.h>
#include <time.h>

/***************************** Equates *****************************/
#define     NULLCHAR     '\0'         /* Null character            */

#define     MATCH         0          /* Comparison status: Equal   */
#define     NOT_OK        0          /* Run status indicator: Error*/
#define     OK            1          /* Run status indicator: Good */

/*********************** DSN8EUMN functions ************************/

/*******************************************************************/
/*************************** main routine **************************/
/*******************************************************************/
void DSN8EUMN                         /* main routine              */
( char        *ISOdateIn,             /* in: date to look up       */
  char        *monthNameOut,          /* out: ISOdateIn's month name*/
  short int   *niISOdateIn,           /* in: indic var, ISOdateIn   */
  short int   *niMonthNameOut,        /* out: indic var,monthNameOut*/
  char        *sqlstate,              /* out: SQLSTATE              */
  char        *fnName,                /* in: family name of function*/
  char        *specificName,          /* in: specific name of func  */
  char        *message                /* out: diagnostic message    */
)
/*******************************************************************
* Returns the name of the month for the date in isoDate.           *
*                                                                  *
* Assumptions:                                                     *
* - *ISOdateIn        points to a char[11], null-terminated string   *
* - *monthNameOut     points to a char[10], null-terminated string   *
* - *niISOdateIn      points to a short integer                      *
* - *niMonthNameOut   points to a short integer                      *
* - *sqlstate         points to a char[06], null-terminated string   *
* - *fnName           points to a char[138], null-terminated string  *
* - *specificName     points to a char[129], null-terminated string  *
* - *message          points to a char[70], null-terminated string   *
*******************************************************************/
{

  /************************ local variables ************************/
  short int   status = OK;            /* DSN8EUMN run status       */
  struct tm   tmbuff;                 /* buffer for time.h tm struct*/
  char        *rc;                    /* gets strf/ptime return code*/
  char        *isoFormat              /* format of isoDate:         */
              = "%Y-%m-%d";           /* %Y = YYYY, %m = MM, %d = DD*/
  char        *fullMonthName          /* format of fullMonthName    */
              = "%B";                 /* %B = full month name       */


  /*****************************************************************
  * Verify that something has been passed in                       *
  *****************************************************************/
  if( *niISOdateIn != 0 || ( strlen( ISOdateIn ) == 0 ) )
    {
      status = NOT_OK;
      strcpy( message,
              "DSN8EUMN Error: No date entered" );
      strcpy( sqlstate, "38601" );
    }

  /*****************************************************************
  * Convert ISOdateIn to C tm format                               *
  *****************************************************************/
  if( status == OK )
    { rc = strptime( ISOdateIn,isoFormat,&tmbuff );
      if( rc == NULL )                /* Unable to convert ISOdateIn*/
        {
          status = NOT_OK;
          strcpy( message,
                  "DSN8EUMN Error: Input date not valid "
                                  "or not in ISO format" );
          strcpy( sqlstate, "38602" );
        }
    }

  /*****************************************************************
  * Convert the date from C tm format to the locale's full monthname *
  *****************************************************************/
  if( status == OK )
```

```
      { *rc = strftime( monthNameOut,10,fullMonthName,&tmbuff );
      }

    /*****************************************************************
    * If month name was obtained, clear the message buffer and sql-    *
    * state, and unset the SQL null indicator for monthNameOut.        *
    *****************************************************************/
    if( status == OK )
      {
        *niMonthNameOut = 0;
        message[0] = NULLCHAR;
        strcpy( sqlstate,"00000" );
      }
    /*****************************************************************
    * If errors occurred, clear the monthNameOut buffer and set the SQL*
    * NULL indicator.  A diagnostic message and the SQLSTATE have been *
    * set where the error was detected.                                *
    *****************************************************************/
    else
      {
        monthNameOut[0] = NULLCHAR;
        *niMonthNameOut = -1;
      }

    return;
  }  /* end DSN8EUMN */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DLPL

Populates the PSEG_PHOTO (500K BLOB) and BMP_PHOTO (100K BLOB) columns of the EMP_PHOTO_RESUME sample table with data read from sequential data sets.

```
  /*******************************************************************
  * Module name = DSN8DLPL (DB2 sample program)                       *
  *                                                                   *
  * DESCRIPTIVE NAME = Populate LOB columns that exceed 32K with data *
  *                    read from sequential data sets.                *
  *                                                                   *
  *                                                                   *
  *  LICENSED MATERIALS - PROPERTY OF IBM                             *
  *  5655-DB2                                                         *
  *  (C) COPYRIGHT 1997 IBM CORP.  ALL RIGHTS RESERVED.               *
  *                                                                   *
  *  STATUS = VERSION 6                                               *
  *                                                                   *
  * Function: Populates the PSEG_PHOTO (500K BLOB) and BMP_PHOTO (100K *
  *           BLOB) columns of the EMP_PHOTO_RESUME sample table with *
  *           data read from sequential data sets.                    *
  *                                                                   *
  *           LOB locators are used to avoid having to contain all the *
  *           data in the application's storage.                      *
  *                                                                   *
  * Notes:                                                            *
  *   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher      *
  *                                                                   *
  *   Restrictions:                                                   *
  *                                                                   *
  * Module type: C program                                            *
  *   Processor: IBM C/C++ for OS/390 V1R3 or subsequent release      *
  * Module size: See linkedit output                                  *
  *  Attributes: Re-entrant and re-usable                             *
  *                                                                   *
  * Entry Point: CEESTART (Language Environment entry point)          *
  *     Purpose: See Function                                         *
  *     Linkage: Standard MVS program invocation, no parameters       *
  *                                                                   *
  *       Input: Symbolic label/name = PSEGINnn, where 00 <= nn <= 99 *
  *              Description = PSEG photo image data                  *
  *                                                                   *
  *              Symbolic label/name = BMPINnn, where 00 <= nn <= 99  *
  *              Description = BMP photo image data                   *
  *                                                                   *
  *      Output: Symbolic label/name = SYSPRINT                       *
  *              Description = Report and messages                    *
```

```
*                                                                     *
*                                                                     *
* Normal Exit: Return Code = 0000                                     *
*              - Message: none                                        *
*                                                                     *
*  Error Exit: Return Code = 0008                                     *
*              - Message: *** ERROR: DSN8DLPL DB2 Sample Program      *
*                                    Unable to open BMPINnn DD data   *
*                                    set.  Processing terminated.     *
*                                                                     *
*              - Message: *** ERROR: DSN8DLPL DB2 Sample Program      *
*                                    Unexpected SQLCODE encountered   *
*                                    at location xxx                  *
*                                    Error detailed below             *
*                                    Processing terminated            *
*                                    (DSNTIAR-formatted message       *
*                                    follows).                        *
*                                                                     *
*    External References:                                             *
*             - Routines/Services: DSNTIAR                            *
*             - Data areas       : DSNTIAR error_message              *
*             - Control blocks   : None                               *
*                                                                     *
*                                                                     *
*  Pseudocode:                                                        *
*   DSN8DLPL:                                                         *
*   - Set DD counter (nn) to 00                                       *
*   - Do while more PSEGINnn DD's to process                          *
*     - Call openPSEGfile to open the data set associated with        *
*       DD PSEGINnn                                                   *
*     - Call getPSEGrec to read the first record of the data set      *
*       - Extract the employee serial from this record                *
*     - Call openBMPfile to open the data set associated with         *
*       DD BMPINnn                                                    *
*     - Call getBMPrec to read the first record of the data set       *
*     - Call primeBLOBcols to:                                        *
*       (a) UPDATE the PSEG_PHOTO and BMP_PHOTO columns of the        *
*           employee's row in the EMP_PHOTO_RESUME table with the     *
*           contents of these first records                           *
*       (b) SELECT the PSEG_PHOTO and BMP_PHOTO columns back into     *
*           BLOB locators                                             *
*     - Call getPSEGrec to read the next record from the PSEGINnn DD *
*     - Do while not end of file for the PSEGINnn DD                  *
*       - Call buildPSEGcol to append the current PSEGINnn record to *
*         the PSEG BLOB locator                                       *
*       - Call getPSEGrec to read the next record from PSEGINnn       *
*     - Call getBMPrec to read the next record from the BMPINnn DD    *
*     - Do while not end of file for the BMPINnn DD                   *
*       - Call buildBMPcol to append the current BMPINnn record to    *
*         the BMP BLOB locator                                        *
*       - Call getBMPrec to read the next record from BMPINnn         *
*     - Call updateBLOBcols to apply the BLOB locators to the         *
*       PSEG_PHOTO and BMP_PHOTO columns of the employee's row in     *
*       the EMP_PHOTO_RESUME table                                    *
*     - If all went well, call commitWorkUnit to commit the changes   *
*     - Else call rollbackWorkUnit to roll back the changes           *
*     - Print a status line                                           *
*     - Close the PSEGINnn and BMPINnn DD's                           *
*     - Increment DD counter (nn) by 1.                               *
*   - If an SQL error occurs, invoke the sql_error routine to gener- *
*     ate and display message text                                    *
*   End DSN8DLPL                                                      *
*                                                                     *
*   openPSEGfile:                                                     *
*   - Open the data set associated with the PSEGINnn DD               *
*   - If the open fails, set validDD to false                         *
*   End openPSEGfile                                                  *
*                                                                     *
*   getPSEGrec:                                                       *
*   - Read a record from the data set associated with the PSEGINnn DD*
*   - If end of file, set morePSEGrecs to false                       *
*   End getPSEGrec                                                    *
*                                                                     *
*   openBMPfile:                                                      *
*   - Open the data set associated with the BMPINnn DD                *
*   End openBMPfile                                                   *
*                                                                     *
*   getBMPrec:                                                        *
*   - Read a record from the data set associated with the BMPINnn DD *
*   - If end of file, set moreBMPrecs to false                        *
*   End getBMPrec                                                     *
*                                                                     *
```

```
*   primeBLOBcols:                                                   *
*   - extract the employee serial from bytes 10-15 of the PSEG       *
*     buffer.                                                        *
*   - UPDATE the PSEG_PHOTO and BMP_PHOTO columns for the employee's *
*     row in the EMP_PHOTO_RESUME table from the PSEG and BMP records*
*   - SELECT the PSEG_PHOTO and BMP_PHOTO columns for the employee   *
*     into LOB locators blPSEG1 and blBMP1
*   End primeBLOBcols                                                *
*                                                                    *
*   buildPSEGcol:                                                    *
*   - append the contents of the PSEG input record to the PSEG BLOB  *
*     locator blPSEG1 and assign to BLOB locator blPSEG2             *
*   - free BLOB locator blPSEG1                                      *
*   - set BLOB locator blPSEG1 from BLOB locator blPSEG2             *
*   - free BLOB locator blPSEG2                                      *
*   End buildPSEGcol                                                 *
*                                                                    *
*   buildBMPcol:                                                     *
*   - append the contents of the BMP input record to the BMP BLOB    *
*     locator blBMP1 and assign to BLOB locator blBMP2               *
*   - free BLOB locator blBMP1                                       *
*   - set BLOB locator blBMP1 from BLOB locator blBMP2               *
*   - free BLOB locator blBMP2                                       *
*   End buildBMPcol                                                  *
*                                                                    *
*   updateBLOBcols:                                                  *
*   - UPDATE the PSEG_PHOTO and BMP_PHOTO columns for the employee's *
*     row in the EMP_PHOTO_RESUME table from the PSEG and BMP BLOB   *
*     locators BMP blPSEG1 and blBMP1                                *
*   End updateBLOBcols                                               *
*                                                                    *
*   commitWorkUnit:                                                  *
*   - commit the changes                                             *
*   End commitWorkUnit                                               *
*                                                                    *
*   rollbackWorkUnit:                                                *
*   - roll back the changes                                          *
*   End rollbackWorkUnit                                             *
*                                                                    *
*   sql_error:                                                       *
*   - call DSNTIAR to format the unexpected SQLCODE.                 *
*   End sql_error                                                    *
*                                                                    *
**********************************************************************/
/********************** C library definitions ***********************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


/***************************** Equates ******************************/
#define       NO             0        /* False                       */
#define       YES            1        /* True                        */

#define       NOT_OK         0        /* Run status indicator: Error*/
#define       OK             1        /* Run status indicator: Good */

#define       TIAR_DIM      10        /* Max no. of DSNTIAR msgs    */
#define       TIAR_LEN      80        /* Length of DSNTIAR messages */


/****************************** Files *******************************/
FILE          *BMPin;                 /* pointer to BMP input file  */
FILE          *PSEGin;                /* pointer to PSEG input file */


/************************* Global Storage *************************/
int           status       = OK;      /* run status flag            */

char          PSEGinDD[12];           /* PSEGin DD template         */
char          BMPinDD[12];            /* BMPin DD template          */
short int     DDcounter    = 0;       /* DD allocation counter      */
char          DDnum[2];               /* DD number string template  */
short int     validDD      = YES;     /* unprocessed DD indicator   */

short int     morePSEGrecs = YES;     /* eof indicator for PSEGINnn */
short int     moreBMPrecs  = YES;     /* eof indicator for BMPINnn  */

short int     PSEGblkLen   = 0;       /* length of PSEG input block */
short int     PSEGblkPos   = 0;       /* offset in PSEG input block */
short int     PSEGrecLen   = 0;       /* length of PSEG input record*/
short int     PSEGrecPos   = 0;       /* offset in PSEG input record*/
```

```
long  int     PSEGcolLen   = 0;       /* length of PSEG column data */

short int     BMPblkLen    = 0;       /* length of BMP input block  */
short int     BMPblkPos    = 0;       /* offset in BMP input block  */
short int     BMPrecLen    = 0;       /* length of BMP input record */
short int     BMPrecPos    = 0;       /* offset in BMP input record */
long  int     BMPcolLen    = 0;       /* length of BMP column data  */

int           byteIn;                 /* current incoming byte      */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/**************************** DB2 Tables ***************************/
EXEC SQL DECLARE  EMP_PHOTO_RESUME  TABLE
  (        EMPNO  CHAR(06)     NOT NULL,
       EMP_ROWID  ROWID,
      PSEG_PHOTO  BLOB( 500K ),
       BMP_PHOTO  BLOB( 100K ),
          RESUME  CLOB(   5K )         );


/************** DB2 Host and Null Indicator Variables ***************/
EXEC SQL BEGIN DECLARE SECTION;
  char        hvEMPNO[7];             /* Host var for employee no.  */

  SQL TYPE IS BLOB(8K) PSEGinRec;     /* Area for PSEG input record */
  short int   niPSEG_PHOTO = 0;       /* Null ind for PSEG photo col*/

  SQL TYPE IS BLOB(8K) BMPinRec;      /* Area for BMP input record  */
  short int   niBMP_PHOTO  = 0;       /* Null ind for BMP photo col */
EXEC SQL END DECLARE SECTION;


/******************** DB2 LOB Locator Variables ********************/
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR blPSEG1;   /* BLOB loc for PSEG photo col*/
  SQL TYPE IS BLOB_LOCATOR blPSEG2;   /* BLOB loc for PSEG photo col*/
  SQL TYPE IS BLOB_LOCATOR blBMP1;    /* BLOB loc for BMP photo col */
  SQL TYPE IS BLOB_LOCATOR blBMP2;    /* BLOB loc for BMP photo col */
EXEC SQL END DECLARE SECTION;


/********************** DB2 Message Formatter **********************/
struct      error_struct {            /* DSNTIAR message structure  */
  short int   error_len;
  char        error_text[TIAR_DIM][TIAR_LEN];
  }         error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage( dsntiar, OS )

extern  short int dsntiar( struct      sqlca         *sqlca,
                           struct      error_struct  *msg,
                           int                       *len );


/************************* Global Functions ************************/
int  main( void );                    /* main routine               */
void openPSEGfile( void );            /* open PSEGINnn DD file      */
void getPSEGrec( void );              /* read PSEG image file       */
void openBMPfile( void );             /* open BMPINnn DD file       */
void getBMPrec( void );               /* read BMP image file        */
void primeBLOBcols( void );           /* set PSEG and BMP BLOB locs */
void buildPSEGcol( void );            /* add to PSEG BLOB locator   */
void buildBMPcol( void );             /* add to BMP BLOB locator    */
void updateBLOBcols( void );          /* apply PSEG and BMP locs    */
void commitWorkUnit( void );          /* commit changes             */
void rollbackWorkUnit( void );        /* roll back changes          */
void sql_error( char *locmsg );       /* generate msg for SQL error */

/*******************************************************************/
/************************** main routine **************************/
/*******************************************************************/
int main( void )
{
  /*****************************************************************/
  /************************ Initialization ************************/
  /*****************************************************************/

  /*****************************************************************
```

```
    * Write identification header                                    *
    ******************************************************************/
    printf( "****************************************"
            "****************************************\n" );
    printf( "* DSN8DLPL DB2 Sample Program\n"             );
    printf( "****************************************"
            "****************************************\n" );

    /******************************************************************/
    /*************************** Processing ***************************/
    /******************************************************************/

    /******************************************************************
    * Cycle through PSEGINnn and BMPINnn DD pairs, incrementing the DD *
    * counter, until all pairs have been processed.                    *
    ******************************************************************/
    for( DDcounter=0; validDD == YES  &&  status == OK; DDcounter++ )
      {
        /**************************************************************
        * Fetch the PSEG data from the current PSEGINnn DD.  The first *
        * record contains the serial number of the employee associated *
        * with the photo.                                              *
        **************************************************************/
        openPSEGfile();
        if( validDD == YES  &&  status == OK )
          getPSEGrec();

        /**************************************************************
        * Fetch the first record from the BMPINnn DD                   *
        **************************************************************/
        if( validDD == YES  &&  status == OK )
          openBMPfile();
        if( validDD == YES  &&  status == OK )
          getBMPrec();

        /**************************************************************
        * Init the PSEG and BMP BLOB table columns for the employee    *
        **************************************************************/
        if( validDD == YES  &&  status == OK )
          primeBLOBcols();

        /**************************************************************
        * Read the second records from the PSEG and BMP data sets      *
        **************************************************************/
        if( validDD == YES  &&  status == OK )
          getPSEGrec();
        if( validDD == YES  &&  status == OK )
          getBMPrec();

        /**************************************************************
        * Append remaining PSEG recs to PSEG photo col using BLOB loc  *
        **************************************************************/
        while( morePSEGrecs == YES && validDD == YES && status == OK )
          {
            buildPSEGcol();
            if( status == OK )
              getPSEGrec();
          }

        /**************************************************************
        * Append remaining BMP recs to BMP photo col using BLOB locator*
        **************************************************************/
        while( moreBMPrecs == YES && validDD == YES && status == OK )
          {
            buildBMPcol();
            if( status == OK )
              getBMPrec();
          }

        /**************************************************************
        * Apply the data associated with the PSEG and BMP BLOB locators*
        * to the table                                                 *
        **************************************************************/
        if( validDD == YES  &&  status == OK )
          updateBLOBcols();

        /**************************************************************
        * If clear status, commit the work unit; otherwise, rollback   *
        **************************************************************/
        if( validDD == YES )
          if( status == OK )
            commitWorkUnit();
```

```
        else
          rollbackWorkUnit();

      /*****************************************************************
      * Print report line                                              *
      *****************************************************************/
      if( validDD == YES  &&  status == OK )
        {
          printf( "*   LOB population statistics for employee "
                  "number %s follow:\n", hvEMPNO );
          printf( "*   - PSEG photo bytes: %d\n",PSEGcolLen );
          printf( "*   - BMP photo bytes: %d\n",BMPcolLen );
          printf( "****************************************"
                  "****************************************\n" );
        }

      /*****************************************************************
      * Close data sets for current PSEGINnn and BMPINnn DDs          *
      *****************************************************************/
      fclose(PSEGin);
      fclose(BMPin);
    } /* end for( DDcounter=0; validDD == YES  &&  status == OK ... */


  /*******************************************************************/
  /***************************** Cleanup ***************************/
  /*******************************************************************/

  /*******************************************************************
  * Set return code                                                  *
  *******************************************************************/
  if( status == OK )
    return( 0 );
  else
    return( 8 );

} /* end main */


void openPSEGfile( void )
/*********************************************************************
* Opens the data set associated with the PSEGINnn DD, where "nn" is  *
* the current setting of the DD counter from the main loop.          *
*                                                                    *
* If the DD cannot be allocated, then no further data sets remain to *
* be processed so signal end of job.                                 *
*********************************************************************/
{
  /*******************************************************************
  * intialize work variables                                         *
  *******************************************************************/
  morePSEGrecs = YES;
  PSEGblkPos = 0;
  PSEGblkLen = 0;

  /*******************************************************************
  * form the DD name for the next PSEG data set                      *
  *******************************************************************/
  strcpy( PSEGinDD,"DD:PSEGIN\0" );   /* init PSEGin DD template    */
  sprintf( DDnum,"%02d",DDcounter );  /* convert DD cntr to string  */
  strcat( PSEGinDD,DDnum );           /* form PSEGINnn DD name      */

  /*******************************************************************
  * open the PSEGINnn DD data set                                    *
  *******************************************************************/
  PSEGin = fopen( PSEGinDD,"rb,recfm=u" );

  if( PSEGin == NULL )                /* if no pointer returned     */
    validDD = NO;                     /* .. no more data sets left  */
} /* end openPSEGfile */


void getPSEGrec( void )
/*********************************************************************
* Called by the main routine to read the next record from the data   *
* set associated with the current PSEGINnn DD into a buffer, PSEGin- *
* Rec.                                                               *
*                                                                    *
* If this is the first record from the PSEGINnn DD data set, it con- *
* tains the serial number of an employee in bytes 10-15 and it will  *
* be UPDATEd into the PSEG_PHOTO column of that employee's row in    *
* the sample EMP_PHOTO_RESUME table.  This column and row will then  *
```

```
* be SELECTed into a BLOB locator, blPSEG1, to be used for accumu-   *
* lating the remaining records from the current PSEGINnn DD data set *
* to form a complete PSEG_PHOTO entry for the current employee.      *
*                                                                    *
* If this is not the first record from the PSEGINnn DD data set, it  *
* will be appended to previously read records for this data set in   *
* a DB2 data area associated with the BLOB locator, blPSEG1.         *
*                                                                    *
* When all records of the data set have been read and accumulated in *
* the locator area, the locator will be applied to the PSEG_PHOTO    *
* column of the current employee's row in the EMP_PHOTO_RESUME table.*
**********************************************************************
* Because the C language is not record-oriented in the sense of MVS  *
* data sets, it's necessary to treat the PSEG data set, which has a  *
* variable-blocked format, as an unformatted dataset in order to     *
* access the block descriptor word (BDW) of each input block and the *
* record descriptor word (RDW) of each input record.                 *
*                                                                    *
* Each RDW provides the number of bytes of data in its record,       *
* including 4 bytes for itself.                                      *
*                                                                    *
* Each BDW provides the number of bytes of data in its block,        *
* including 4 bytes for each RDW in the block and 4 bytes for        *
* itself.                                                            *
*********************************************************************/
{
  /*******************************************************************
  * intialize work variables                                         *
  *******************************************************************/
  PSEGrecLen = 0;
  PSEGrecPos = 0;
  PSEGinRec.length = 0;

  /*******************************************************************
  * read the 1st byte of the record                                  *
  *******************************************************************/
  byteIn = getc(PSEGin);
  /*******************************************************************
  * get remaining bytes of the record if not EOF                     *
  *******************************************************************/
  if( byteIn != EOF )
    {
      /***************************************************************
      * if at end of block, read next BDW                            *
      ***************************************************************/
      if( PSEGblkPos >= PSEGblkLen  &&  PSEGrecPos >= PSEGrecLen)
        {
          /***********************************************************
          * length of block = (16**2) * BDW[0]                       *
          * .............. + (16**0) * BDW[1]                        *
          * .............. - 4 (length of BDW)                       *
          ***********************************************************/
          PSEGblkLen = 256 * byteIn;
          byteIn = getc(PSEGin);
          PSEGblkLen = PSEGblkLen + byteIn - 4;

          /***********************************************************
          * skip remainder of BDW                                    *
          ***********************************************************/
          byteIn = getc(PSEGin);
          byteIn = getc(PSEGin);
          PSEGblkPos = 0;

          /***********************************************************
          * read first byte of RDW                                   *
          ***********************************************************/
          byteIn = getc(PSEGin);
        }
      /***************************************************************
      * process the RDW                                              *
      ****************************************************************
      * length of record = (16**2) * RDW[0]                          *
      * ............... + (16**0) * RDW[1]                           *
      * ............... - 4 (length of RDW)                          *
      ***************************************************************/
      PSEGrecLen = 256 * byteIn;
      byteIn = getc(PSEGin);
      PSEGrecLen = PSEGrecLen + byteIn - 4;

      /***************************************************************
      * skip remainder of RDW                                        *
      ***************************************************************/
```

```
      byteIn = getc(PSEGin);
      byteIn = getc(PSEGin);
      PSEGrecPos = 0;

      /*****************************************************************
      * update position in block                                        *
      *****************************************************************/
      PSEGblkPos = PSEGblkPos + PSEGrecLen + 4;
    }

  /*********************************************************************
  * build the PSEG record according to the record length               *
  *********************************************************************/
  while( PSEGrecPos < PSEGrecLen  &&  byteIn != EOF )
    {
      byteIn = getc(PSEGin);
      PSEGinRec.data[PSEGinRec.length++] = byteIn;
      PSEGrecPos++;
    }

  /*********************************************************************
  * signal end of file when applicable                                 *
  *********************************************************************/
  if( byteIn == EOF )
    morePSEGrecs = NO;

} /* end getPSEGrec */


void openBMPfile( void )
/***********************************************************************
* Opens the data set associated with the BMPINnn DD, where "nn" is   *
* the current setting of the DD counter from the main loop.          *
*                                                                      *
* If the DD cannot be allocated, then an error has occurred because  *
* each BMPINnn DD must be paired with a PSEGINnn data set.           *
***********************************************************************/
{
  /*********************************************************************
  * intialize work variables                                           *
  *********************************************************************/
  moreBMPrecs = YES;
  BMPblkPos = 0;
  BMPblkLen = 0;

  /*********************************************************************
  * form the DD name for the next BMP data set                         *
  *********************************************************************/
  strcpy( BMPinDD,"DD:BMPIN\0" );     /* init BMPin DD template     */
  sprintf( DDnum,"%02d",DDcounter );  /* convert DD cntr to string  */
  strcat( BMPinDD,DDnum );            /* form BMPINnn DD name       */

  /*********************************************************************
  * open the current BMPINnn DD data set                               *
  *********************************************************************/
  BMPin = fopen( BMPinDD,"rb,recfm=u" );

  if( BMPin == NULL )
    {
      printf( "************************************************\n" );
      printf( "*** ERROR: DSN8DLPL DB2 Sample Program\n"           );
      printf( "***        Unable to open BMPIN%s DD data set\n",
              DDnum );
      printf( "***        Processing terminated.\n"               );
      printf( "************************************************\n" );
      status = NOT_OK;
    }
} /* end openBMPfile */


void getBMPrec( void )
/***********************************************************************
* Called by the main routine to read the next record from the data   *
* set associated with the current BMPINnn DD into a buffer, BMPinRec.*
*                                                                      *
* If this is the first record from the BMPINnn DD data set, it con-  *
* tains the serial number of an employee in bytes 10-15 and it will  *
* be UPDATEd into the BMP_PHOTO column of that employee's row in     *
* the sample EMP_PHOTO_RESUME table.  This column and row will then  *
* be SELECTed into a BLOB locator, blBMP1, to be used for accumulat- *
* ing the remaining records from the current BMPINnn DD data set to  *
* form a complete BMP_PHOTO entry for the current employee.          *
```

```
*                                                                      *
* If this is not the first record from the BMPINnn DD data set, it     *
* will be appended to previously read records for this data set in     *
* a DB2 data area associated with the BLOB locator, blBMP1.            *
*                                                                      *
* When all records of the data set have been read and accumulated in *
* the locator area, the locator will be applied to the BMP_PHOTO       *
* column of the current employee's row in the EMP_PHOTO_RESUME table.*
**********************************************************************
* Because the C language is not record-oriented in the sense of MVS  *
* data sets, it's necessary to treat the BMP data set, which has a    *
* variable-blocked format, as an unformatted dataset in order to      *
* access the block descriptor word (BDW) of each input block and the *
* record descriptor word (RDW) of each input record.                  *
*                                                                      *
* Each RDW provides the number of bytes of data in its record,        *
* including 4 bytes for itself.                                        *
*                                                                      *
* Each BDW provides the number of bytes of data in its block,         *
* including 4 bytes for each RDW in the block and 4 bytes for         *
* itself.                                                              *
*********************************************************************/
{
  /*******************************************************************
  * intialize work variables                                         *
  *******************************************************************/
  BMPrecLen = 0;
  BMPrecPos = 0;
  BMPinRec.length = 0;

  /*******************************************************************
  * read the 1st byte of the record                                  *
  *******************************************************************/
  byteIn = getc(BMPin);
  /*******************************************************************
  * get remaining bytes of the record if not EOF                     *
  *******************************************************************/
  if( byteIn != EOF )
    {
      /***************************************************************
      * if at end of block, read next BDW                            *
      ***************************************************************/
      if( BMPblkPos >= BMPblkLen )
        {
          /***********************************************************
          * length of block = (16**2) * BDW[0]                       *
          * .............. + (16**0) * BDW[1]                        *
          * .............. - 4 (length of BDW)                      *
          ***********************************************************/
          BMPblkLen = 256 * byteIn;
          byteIn = getc(BMPin);
          BMPblkLen = BMPblkLen + byteIn - 4;

          /***********************************************************
          * skip remainder of BDW                                    *
          ***********************************************************/
          byteIn = getc(BMPin);
          byteIn = getc(BMPin);
          BMPblkPos = 0;

          /***********************************************************
          * read first byte of RDW                                   *
          ***********************************************************/
          byteIn = getc(BMPin);
        }
      /***************************************************************
      * process the RDW                                              *
      ****************************************************************
      * length of record = (16**2) * RDW[0]                          *
      * ............... + (16**0) * RDW[1]                           *
      * ............... - 4 (length of RDW)                          *
      ***************************************************************/
      BMPrecLen = 256 * byteIn;
      byteIn = getc(BMPin);
      BMPrecLen = BMPrecLen + byteIn - 4;

      /***************************************************************
      * skip remainder of RDW                                        *
      ***************************************************************/
      byteIn = getc(BMPin);
      byteIn = getc(BMPin);
      BMPrecPos = 0;
```

```
    /************************************************************
    * update position in block                                  *
    ************************************************************/
    BMPblkPos = BMPblkPos + BMPrecLen + 4;
  }

  /****************************************************************
  * build the BMP record according to the record length           *
  ****************************************************************/
  while( BMPrecPos < BMPrecLen     &&  byteIn != EOF )
    {
    byteIn = getc(BMPin);
    BMPinRec.data[BMPinRec.length++] = byteIn;
    BMPrecPos++;
    }

  /****************************************************************
  * signal end of file when applicable                            *
  ****************************************************************/
  if( byteIn == EOF )
    moreBMPrecs = NO;

} /* end getBMPrec */


void primeBLOBcols( void )
/*******************************************************************
* Called by the main routine to apply the first PSEG input record    *
* (from getPSEGrec) and the first BMP input record (from getBMPrec)  *
* to the PSEG_PHOTO and BMP_PHOTO BLOB columns, respectively, and    *
* then fetch those columns using BLOB locators.                      *
*                                                                    *
* The PSEG BLOB locator will be used by the buildPSEGcol function    *
* to build a BLOB entity of up to 500K bytes from the remaining      *
* PSEGin records without consuming application workspace.           *
*                                                                    *
* The BMP BLOB locator will be used by the buildBMPcol function to   *
* build a BLOB entity of up to 500K bytes from the remaining BMPin   *
* records, again without consuming application workspace.            *
*                                                                    *
* When all PSEG and BMP records have been processed, the data will   *
* be applied from the BLOB locators to the EMP_PHOTO_RESUME table by *
* the updateBLOBcols function.                                       *
*******************************************************************/
{
  char *empser;                        /*                            */

  /****************************************************************
  * Extract the employee number from bytes 10-15 of the PSEG record *
  ****************************************************************/
  empser = &PSEGinRec.data[9];
  strncpy( hvEMPNO,empser,6 );

  /****************************************************************
  * Initialize the BLOB columns with data from the 1st input records *
  ****************************************************************/
  EXEC SQL UPDATE  EMP_PHOTO_RESUME
             SET  PSEG_PHOTO = :PSEGinRec,
                  BMP_PHOTO  = :BMPinRec
           WHERE  EMPNO = :hvEMPNO;

  if( SQLCODE != 0 )
    {
    status = NOT_OK;
    sql_error( "primeBLOBcols @ UPDATE" );
    }

  /****************************************************************
  * Select the initial BLOB data into locators                    *
  ****************************************************************/
  if( status == OK )
    {
    EXEC SQL SELECT  PSEG_PHOTO, BMP_PHOTO
               INTO :blPSEG1 :niPSEG_PHOTO,
                    :blBMP1  :niBMP_PHOTO
               FROM  EMP_PHOTO_RESUME
              WHERE  EMPNO = :hvEMPNO;

      if( SQLCODE != 0 )
        {
        status = NOT_OK;
```

```
        sql_error( "primeBLOBcols @ SELECT" );
      }

    }

  /*****************************************************************
  * Set initial lengths of PSEG_PHOTO anf BMP_PHOTO columns          *
  *****************************************************************/
  PSEGcolLen = PSEGinRec.length;
  BMPcolLen  = BMPinRec.length;

} /* end primeBLOBcols */


void buildPSEGcol( void )
/*******************************************************************
* Called by the main routine to build a PSEG_PHOTO column entry for  *
* the current employee.                                              *
*                                                                    *
* This is done by appending the current record of the PSEG input file*
* (from getPSEGrec) to the entity associated with blPSEG1, the BLOB  *
* locator for the PSEG_PHOTO column.                                 *
*                                                                    *
* When all PSEG input records have been appended to this entity, the *
* updateBLOBcols function will be invoked to update the PSEG_PHOTO   *
* column in the EMP_PHOTO_RESUME table from blPSEG1.                 *
*******************************************************************/
{
  /*****************************************************************
  * Generate a new BLOB locator that contains the current input      *
  * record appended to the current PSEG_PHOTO locator                *
  *****************************************************************/
  EXEC SQL SET  :blPSEG2 = SUBSTR( :blPSEG1,1,LENGTH(:blPSEG1) )
                     ||  :PSEGinRec;

  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "buildPSEGcol @ SET LOCATOR #2" );
    }

  /*****************************************************************
  * Regenerate the PSEG_PHOTO locator from the updated locator       *
  *****************************************************************/
  if( status == OK )
    {
      EXEC SQL FREE LOCATOR :blPSEG1;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildPSEGcol @ FREE LOCATOR #1" );
        }
    }

  if( status == OK )
    {
      EXEC SQL SET :blPSEG1 = :blPSEG2;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildPSEGcol @ SET LOCATOR #1" );
        }
    }

  if( status == OK )
    {
      EXEC SQL FREE LOCATOR :blPSEG2;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildPSEGcol @ FREE LOCATOR #2" );
        }
    }

  /*****************************************************************
  * Update length of PSEG_PHOTO column                               *
  *****************************************************************/
  if( status == OK )
    PSEGcolLen = PSEGcolLen + PSEGinRec.length;

} /* end buildPSEGcol */
```

```
void buildBMPcol( void )
/**********************************************************************
* Called by the main routine to build a BMP_PHOTO column entry for   *
* the current employee.                                              *
*                                                                    *
* This is done by appending the current record of the BMP input file *
* (from getBMPrec) to the entity associated with blBMP1, the BLOB    *
* locator for the BMP_PHOTO column.                                  *
*                                                                    *
* When all BMP input records have been appended to this entity, the  *
* updateBLOBcols function will be invoked to update the BMP_PHOTO    *
* column in the EMP_PHOTO_RESUME table from blBMP1.                  *
**********************************************************************/
{
  /*******************************************************************
  * Generate a new BLOB locator that contains the current input      *
  * record appended to the current BMP_PHOTO locator                 *
  *******************************************************************/
  EXEC SQL SET  :blBMP2 = SUBSTR( :blBMP1,1,LENGTH(:blBMP1) )
                            ||  :BMPinRec;

  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "buildBMPcol @ SET LOCATOR #2" );
    }

  /*******************************************************************
  * Regenerate the BMP_PHOTO locator from the updated locator        *
  *******************************************************************/
  if( status == OK )
    {
      EXEC SQL FREE LOCATOR :blBMP1;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildBMPcol @ FREE LOCATOR #1" );
        }
    }

  if( status == OK )
    {
      EXEC SQL SET :blBMP1 = :blBMP2;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildBMPcol @ SET LOCATOR #1" );
        }
    }

  if( status == OK )
    {
      EXEC SQL FREE LOCATOR :blBMP2;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "buildBMPcol @ FREE LOCATOR #2" );
        }
    }

  /*******************************************************************
  * Update length of BMP_PHOTO column                                *
  *******************************************************************/
  if( status == OK )
    BMPcolLen = BMPcolLen + BMPinRec.length;

} /* end buildBMPcol */


void updateBLOBcols( void )
/**********************************************************************
* Called by the main routine to apply the BLOB entities constructed  *
* from the PSEGin and BMPin input files by the buildPSEGcol and      *
* buildBMPcol functions and pointed to by the blPSEG1 and blBMP1     *
* BLOB locators to the PSEG_PHOTO and BMP_PHOTO columns of the       *
* EMP_PHOTO_RESUME_TABLE.                                            *
**********************************************************************/
{
  EXEC SQL UPDATE  EMP_PHOTO_RESUME
             SET  PSEG_PHOTO = :blPSEG1,
                  BMP_PHOTO  = :blBMP1
```

```
            WHERE  EMPNO = :hvEMPNO;

  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "updateBLOBcols @ UPDATE" );
    }
} /* end updateBLOBcols */


void commitWorkUnit( void )
/**********************************************************************
* Called by the main routine to commit the current unit of work,     *
* which is composed of a fully-built PSEG entry and a fully-built    *
* BMP entry for the current employee.                                *
**********************************************************************/
{
  EXEC SQL COMMIT;

  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "commitWorkUnit @ COMMIT" );
    }
} /* end commitWorkUnit */


void rollbackWorkUnit( void )
/**********************************************************************
* Called by the main routine to rollback the current unit of work,   *
* which is composed of a fully-built PSEG entry and a fully-built    *
* BMP entry for the current employee.                                *
**********************************************************************/
{
  EXEC SQL ROLLBACK;

  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "rollbackWorkUnit @ ROLLBACK" );
    }
} /* end rollbackWorkUnit */


void sql_error( char *locmsg )
/**********************************************************************
* SQL error handler                                                  *
**********************************************************************/
{
  short int   rc;                    /* DSNTIAR Return code        */
  int         j,k;                   /* Loop control               */
  static int  lrecl = TIAR_LEN;      /* Width of message lines     */

  /********************************************************************
  * print the location message                                       *
  ********************************************************************/
  printf( "*****************************************************\n" );
  printf( "*** ERROR: DSN8DLPL DB2 Sample Program\n"                );
  printf( "***        Unexpected SQLCODE encountered at location\n" );
  printf( "***         %.68s\n", locmsg                             );
  printf( "***        Error detailed below\n"                       );
  printf( "***        Processing terminated\n"                      );
  printf( "*****************************************************\n" );

  /********************************************************************
  * format and print the SQL message                                 *
  ********************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<TIAR_DIM; j++ )
      {
        for( k=0; k<TIAR_LEN; k++ )
          putchar(error_message.error_text[j][k] );
        putchar('\n');
      }
  else
    {
      printf( " *** ERROR: DSNTIAR could not format the message\n" );
      printf( " ***        SQLCODE is %d\n",SQLCODE );
      printf( " ***        SQLERRM is \n" );
      for( j=0; j<sqlca.sqlerrml; j++ )
        printf( "%c", sqlca.sqlerrmc[j] );
```

```
      printf( "\n" );
    }

} /* end sql_error */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DLRV

Prompts the user to choose an employee, then retrieves the resume data for that employee from the RESUME (CLOB) column of the EMP_PHOTO_RESUME table into a CLOB locator, uses LOB locator-handling functions to locate and break out data elements, and puts them in fields for display by ISPF.

```
/*********************************************************************
* Module name = DSN8DLRV (DB2 sample program)                       *
*                                                                    *
* DESCRIPTIVE NAME = Display the resume of a specified employee      *
*                                                                    *
*                                                                    *
*  LICENSED MATERIALS - PROPERTY OF IBM                              *
*  5675-DB2                                                          *
*  (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.          *
*                                                                    *
*  STATUS = VERSION 7                                                *
*                                                                    *
* Function: Prompts the user to choose an employee, then retrieves   *
*           the resume data for that employee from the RESUME (CLOB) *
*           column of the EMP_PHOTO_RESUME table into a CLOB locator,*
*           uses LOB locator-handling functions to locate and break  *
*           out data elements, and puts them in fields for display   *
*           by ISPF.                                                 *
*                                                                    *
* Notes:                                                             *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                                                                    *
*   Restrictions:                                                    *
*                                                                    *
* Module type: C program                                             *
*   Processor: IBM C/C++ for OS/390 V1R3 or subsequent release       *
* Module size: See linkedit output                                   *
*  Attributes: Re-entrant and re-usable                              *
*                                                                    *
* Entry Point: CEESTART (Language Environment entry point)           *
*     Purpose: See Function                                          *
*     Linkage: Standard MVS program invocation, no parameters        *
*                                                                    *
* Normal Exit: Return Code = 0000                                    *
*              - Message: none                                       *
*                                                                    *
*  Error Exit: Return Code = 0008                                    *
*              - Message: *** ERROR: DSN8DLRV DB2 Sample Program     *
*                                    Unexpected SQLCODE encountered  *
*                                    at location xxx                 *
*                                    Error detailed below            *
*                                    Processing terminated           *
*                                    (DSNTIAR-formatted message here)*
*                                                                    *
*              - Message: *** ERROR: DSN8DLRV DB2 Sample Program     *
*                                    No entry in the Employee Photo/ *
*                                    Resume table for employee with  *
*                                    empno = xxxxxx                  *
*                                    Processing terminated           *
*                                                                    *
*              - Message: *** ERROR: DSN8DLRV DB2 Sample Program     *
*                                    No resume data exists in        *
*                                    the Employee Photo/Resume table *
*                                    for the employee with empno =   *
*                                    xxxxxx.                         *
*                                    Processing terminated           *
*                                                                    *
*                                                                    *
*    External References:                                            *
*             - Routines/Services: DSNTIAR, ISPF                     *
*             - Data areas       : DSNTIAR error_message             *
*             - Control blocks   : None                              *
```

```
*                                                                    *
*                                                                    *
*  Pseudocode:                                                       *
*   DSN8DLRV:                                                        *
*   - Call initISPFvars to establish ISPF variable sharing           *
*   - Do until the user indicates termination                        *
*     - Call clearISPFvars to reset the ISPF shared variables        *
*     - Call getEmplNum to request an employee id                    *
*     - Call getEmplResume to retrieve the resume                    *
*     - Call formatEmplResume to populate the ISPF display panel     *
*     - Call showEmplResume to display the resume                    *
*   - Call freeISPFvars to terminate ISPF variable sharing           *
*   End DSN8DLRV                                                     *
*                                                                    *
*   initISPFvars:                                                    *
*   - Establish ISPF variable sharing                                *
*   End initISPFvars                                                 *
*                                                                    *
*   clearISPFvars:                                                   *
*   - Set ISPF vars to blank if character type or 0 if numeric       *
*   End clearISPFvars                                                *
*                                                                    *
*   getEmplNum:                                                      *
*   - prompt user to select an employee whose resume is to be viewed *
*   End getEmplNum                                                   *
*                                                                    *
*   getEmplResume:                                                   *
*   - Fetch the specified employee's resume from DB2 using a CLOB    *
*     locator                                                        *
*   End getEmplResume                                                *
*                                                                    *
*   formatEmplResume:                                                *
*   - call getPersonalData to extract personal data from the resume  *
*   - call getDepartmentData to extract department data              *
*   - call getEducationData to extract education data                *
*   - call getWorkHistoryData to extract work history data           *
*   End formatEmplResume                                             *
*                                                                    *
*   showEmplResume:                                                  *
*   - Display the ISPF panel with the specified employee's resume    *
*   End showEmplResume                                               *
*                                                                    *
*   freeISPFvars:                                                    *
*   - Terminate variable sharing with ISPF                           *
*   End freeISPFvars                                                 *
*                                                                    *
*   getPersonalData:                                                 *
*   - Parse the employee's name, address, home telephone no.,        *
*     birthdate, sex, marital status, height, and weight into ISPF   *
*     display variables                                              *
*   End getPersonalData                                              *
*                                                                    *
*   getDepartmentData:                                               *
*   - Parse the employee's department number, manager, job position, *
*     work telephone no., and hire date into ISPF display variables. *
*   End getDepartmentData                                            *
*                                                                    *
*   getEducationData:                                                *
*   - Parse the employee's degree dates, descriptions, and schools   *
*     into ISPF display variables.                                   *
*   End getEducationData                                             *
*                                                                    *
*   getWorkHistoryData:                                              *
*   - Parse the employee's job dates, titles, and descriptions into  *
*     ISPF display variables.                                        *
*   End getWorkHistoryData                                           *
*                                                                    *
*   sql_error:                                                       *
*   - call DSNTIAR to format an unexpected SQLCODE.                  *
*   End sql_error                                                    *
*                                                                    *
**********************************************************************
* Assumptions:                                                       *
* (1) Each employee has exactly 2 entries under "Education"          *
* (2) Each employee has exactly 3 entries under "Work History"       *
* (3) Each job description consists of a single sentence and that    *
*     sentence ends with a period and that period is the only        *
*     period in the sentence.                                        *
*********************************************************************/

/****************** C Program Product Libraries *******************/
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>


/***************************** Equates *****************************/
#define     NO                  0     /* False                        */
#define     YES                 1     /* True                         */

#define     NOT_OK              0     /* Run status indicator: Error*/
#define     OK                  1     /* Run status indicator: Good */

#define     TIAR_DIM           10     /* Max no. of DSNTIAR msgs    */
#define     TIAR_LEN           80     /* Length of DSNTIAR messages */


/************************** Global Storage *************************/
int         keepViewing     = YES;    /* User status                */
int         status          = OK;     /* Run status                 */

short int   ISPFrc;                   /* For ISPF return code       */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/********************** DB2 Message Formatter **********************/
struct      error_struct {            /* DSNTIAR message structure  */
  short int   error_len;
  char        error_text[TIAR_DIM][TIAR_LEN];
  }         error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );


/**************************** DB2 Tables ***************************/
EXEC SQL DECLARE  EMP_PHOTO_RESUME  TABLE
  (        EMPNO  CHAR(06)     NOT NULL,
       EMP_ROWID  ROWID,
      PSEG_PHOTO  BLOB( 500K ),
       BMP_PHOTO  BLOB( 100K ),
          RESUME  CLOB(   5K )         );


/************** DB2 Host and Null Indicator Variables ***************/
EXEC SQL BEGIN DECLARE SECTION;
  char        hvEMPNO[7];             /* host var for emp ser no.   */

  long int    begSection;             /* ptr to beg of resume sec'n */
  char        *begField;              /* ptr to beg of fld in sec'n */
  long int    endSection;             /* ptr to end of resume sec'n */
  char        *endField;              /* ptr to end of fld in sec'n */

  SQL TYPE IS CLOB(5K) hvRESUME;      /* host var for RESUME CLOB   */
  char        *phvRESUME;             /* ptr to RESUME CLOB data    */
  short int   niRESUME = 0;           /* indic var for RESUME CLOB  */
EXEC SQL END DECLARE SECTION;


/******************** DB2 LOB Locator Variables ********************/
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB_LOCATOR clRESUME;  /* CLOB loc for RESUME column */
EXEC SQL END DECLARE SECTION;


/*************************** ISPF Linkage **************************/
#pragma linkage(isplink,OS)


/*************************** ISPF Syntax ***************************/
char        CHAR[9]      = "CHAR    ";
char        DISPLAY[9]   = "DISPLAY ";
char        VDEFINE[9]   = "VDEFINE ";
char        VGET[9]      = "VGET    ";
char        VRESET[9]    = "VRESET  ";


/********************** ISPF Shared Variables **********************/
```

```
char        D8EMNAME[25];            /* employee's name            */
char        D8EMNUMB[7];             /* employee's serial number   */
char        D8EMADR1[25];            /* employee's address line 1  */
char        D8EMDEPT[5];             /* employee's department      */
char        D8EMADR2[25];            /* employee's address line 2  */
char        D8MGRNAM[22];            /* employee's manager's name  */
char        D8EMADR3[15];            /* employee's address line 3  */
char        D8EMPOSN[22];            /* employee's job position    */
char        D8EMBORN[19];            /* employee's date of birth   */
char        D8EMPHON[15];            /* employee's home phone no.  */
char        D8EMSEX[7];              /* employee's gender          */
char        D8EMHIRE[11];            /* employee's hire date       */
char        D8EMHGT[6];              /* employee's height          */
char        D8EMWGT[9];              /* employee's weight          */
char        D8EMPMST[9];             /* employee's marital status  */
char        D8EMEDY1[5];             /* date of most recent degree */
char        D8EMEDD1[35];            /* type of most recent degree */
char        D8EMEDY2[5];             /* date of previous degree    */
char        D8EMEDD2[35];            /* type of previous degree    */
char        D8EMEDI1[35];            /* name of most recent school */
char        D8EMEDI2[35];            /* name of previous school    */
char        D8EMWHD1[17];            /* dates of 1st previous job  */
char        D8EMWHJ1[63];            /* title of 1st previous job  */
char        D8EMWHT1[63];            /* descr. of 1st previous job */
char        D8EMWHD2[17];            /* dates of 2nd previous job  */
char        D8EMWHJ2[63];            /* title of 2nd previous job  */
char        D8EMWHT2[63];            /* descr. of 2nd previous job */
char        D8EMWHD3[17];            /* dates of 3rd previous job  */
char        D8EMWHJ3[63];            /* title of 3rd previous job  */
char        D8EMWHT3[63];            /* descr. of 3rd previous job */


/************************* Global Functions *************************/
int  main( void );                   /* main logic                 */
void initISPFvars( void );           /* establish ISPF vars        */
void clearISPFvars( void );          /* blank/zero ISPF disp vars  */
void getEmplNum( void );             /* prompt for employee ser no */
void getEmplResume( void );          /* get resume from database   */
void formatEmplResume( void );       /* build display panel        */
void getPersonalData( void );        /* get personal data from res */
void getDepartmentData( void );      /* get dept data from resume  */
void getEducationData( void );       /* get educ data from resume  */
void getWorkHistoryData( void );     /* get job hist from resume   */
void showEmplResume( void );         /* display the ISPF panel     */
void freeISPFvars( void );           /* drop ISPF vars             */
void sql_error( char *locmsg );      /* generate SQL messages      */


/*******************************************************************
**************************** main routine ***************************
*******************************************************************/
int main( void )
{
  /******************************************************************
  * Establish variable sharing with ISPF                            *
  ******************************************************************/
  initISPFvars();

  /******************************************************************
  * Display employee resumes until user indicates completion        *
  ******************************************************************/
  keepViewing = YES;
  while( keepViewing == YES )
    {
      clearISPFvars();
      /**************************************************************
      * prompt user to select employee whose resume is to be viewed  *
      **************************************************************/
      getEmplNum();

      if( keepViewing == YES  &&  status == OK )
        {
          /**********************************************************
          * retrieve the employee's resume from DB2                 *
          **********************************************************/
          getEmplResume();
          /**********************************************************
          * if successful, format the resume on ISPF                *
          **********************************************************/
          if( status == OK )
            formatEmplResume();
          /**********************************************************
```

```
            * if successful, display the resume on ISPF                *
            ***********************************************************/
            if( status == OK )
              showEmplResume();
            /***********************************************************
            * otherwise, exit this program                             *
            ***********************************************************/
            else
              keepViewing = NO;
          }
      }

    /*****************************************************************
    * Terminate variable sharing with ISPF                           *
    *****************************************************************/
    freeISPFvars();

  } /* end main */


  void initISPFvars( void )
  /*******************************************************************
  * Called by the main routine. Establishes variable sharing between   *
  * ISPF and this program.                                           *
  *******************************************************************/
  {
    ISPFrc = isplink( VDEFINE, "D8EMNAME", D8EMNAME, CHAR, 24 );
    ISPFrc = isplink( VDEFINE, "D8EMNUMB", D8EMNUMB, CHAR, 6 );
    ISPFrc = isplink( VDEFINE, "D8EMADR1", D8EMADR1, CHAR, 24 );
    ISPFrc = isplink( VDEFINE, "D8EMDEPT", D8EMDEPT, CHAR, 4 );
    ISPFrc = isplink( VDEFINE, "D8EMADR2", D8EMADR2, CHAR, 24 );
    ISPFrc = isplink( VDEFINE, "D8MGRNAM", D8MGRNAM, CHAR, 21 );
    ISPFrc = isplink( VDEFINE, "D8EMADR3", D8EMADR3, CHAR, 14 );
    ISPFrc = isplink( VDEFINE, "D8EMPOSN", D8EMPOSN, CHAR, 21 );
    ISPFrc = isplink( VDEFINE, "D8EMBORN", D8EMBORN, CHAR, 18 );
    ISPFrc = isplink( VDEFINE, "D8EMPHON", D8EMPHON, CHAR, 14 );
    ISPFrc = isplink( VDEFINE, "D8EMSEX ", D8EMSEX , CHAR, 6 );
    ISPFrc = isplink( VDEFINE, "D8EMHIRE", D8EMHIRE, CHAR, 10 );
    ISPFrc = isplink( VDEFINE, "D8EMHGT ", D8EMHGT , CHAR, 5 );
    ISPFrc = isplink( VDEFINE, "D8EMWGT ", D8EMWGT , CHAR, 8 );
    ISPFrc = isplink( VDEFINE, "D8EMPMST", D8EMPMST, CHAR, 8 );
    ISPFrc = isplink( VDEFINE, "D8EMEDY1", D8EMEDY1, CHAR, 4 );
    ISPFrc = isplink( VDEFINE, "D8EMEDD1", D8EMEDD1, CHAR, 34 );
    ISPFrc = isplink( VDEFINE, "D8EMEDY2", D8EMEDY2, CHAR, 4 );
    ISPFrc = isplink( VDEFINE, "D8EMEDD2", D8EMEDD2, CHAR, 34 );
    ISPFrc = isplink( VDEFINE, "D8EMEDI1", D8EMEDI1, CHAR, 34 );
    ISPFrc = isplink( VDEFINE, "D8EMEDI2", D8EMEDI2, CHAR, 34 );
    ISPFrc = isplink( VDEFINE, "D8EMWHD1", D8EMWHD1, CHAR, 16 );
    ISPFrc = isplink( VDEFINE, "D8EMWHJ1", D8EMWHJ1, CHAR, 62 );
    ISPFrc = isplink( VDEFINE, "D8EMWHT1", D8EMWHT1, CHAR, 62 );
    ISPFrc = isplink( VDEFINE, "D8EMWHD2", D8EMWHD2, CHAR, 16 );
    ISPFrc = isplink( VDEFINE, "D8EMWHJ2", D8EMWHJ2, CHAR, 62 );
    ISPFrc = isplink( VDEFINE, "D8EMWHT2", D8EMWHT2, CHAR, 62 );
    ISPFrc = isplink( VDEFINE, "D8EMWHD3", D8EMWHD3, CHAR, 16 );
    ISPFrc = isplink( VDEFINE, "D8EMWHJ3", D8EMWHJ3, CHAR, 62 );
    ISPFrc = isplink( VDEFINE, "D8EMWHT3", D8EMWHT3, CHAR, 62 );
  } /* end initISPFvars */


  void clearISPFvars( void )
  /*******************************************************************
  * Called by the main routine. Blanks out the ISPF shared variables.  *
  *******************************************************************/
  {
    memset( D8EMNAME, 0, 25 );
    memset( D8EMNUMB, 0, 7 );
    memset( D8EMADR1, 0, 25 );
    memset( D8EMDEPT, 0, 5 );
    memset( D8EMADR2, 0, 25 );
    memset( D8MGRNAM, 0, 22 );
    memset( D8EMADR3, 0, 15 );
    memset( D8EMPOSN, 0, 22 );
    memset( D8EMBORN, 0, 19 );
    memset( D8EMPHON, 0, 15 );
    memset( D8EMSEX , 0, 7 );
    memset( D8EMHIRE, 0, 11 );
    memset( D8EMHGT , 0, 9 );
    memset( D8EMWGT , 0, 8 );
    memset( D8EMPMST, 0, 9 );
    memset( D8EMEDY1, 0, 5 );
    memset( D8EMEDD1, 0, 35 );
    memset( D8EMEDY2, 0, 5 );
```

```
  memset( D8EMEDD2, 0, 35 );
  memset( D8EMEDI1, 0, 35 );
  memset( D8EMEDI2, 0, 35 );
  memset( D8EMWHD1, 0, 17 );
  memset( D8EMWHJ1, 0, 63 );
  memset( D8EMWHT1, 0, 63 );
  memset( D8EMWHD2, 0, 17 );
  memset( D8EMWHJ2, 0, 63 );
  memset( D8EMWHT2, 0, 63 );
  memset( D8EMWHD3, 0, 17 );
  memset( D8EMWHJ3, 0, 63 );
  memset( D8EMWHT3, 0, 63 );
} /* end clearISPFvars */


void getEmplNum( void )
/**********************************************************************
* Called by the main routine.  Displays an ISPF panels to prompt the *
* user to select an employee whose resume is to be displayed.        *
**********************************************************************/
{
  /********************************************************************
  * Display the prompt panel                                          *
  ********************************************************************/
  ISPFrc = isplink( "DISPLAY ","DSN8SSE " );
  if( ISPFrc != 0 )
    keepViewing = NO;

  /********************************************************************
  * Save off the value of the ISPF shared variable                    *
  ********************************************************************/
  strcpy( hvEMPNO,D8EMNUMB );

} /* end getEmplNum */


void getEmplResume( void )
/**********************************************************************
* Called by the main routine.  Extracts a specified employee's       *
* resume data from a CLOB column in the sample EMP_PHOTO_RESUME      *
* table to a CLOB locator.                                           *
**********************************************************************/
{
  /********************************************************************
  * Establish a CLOB locator on the resume of the specified empno    *
  ********************************************************************/
  EXEC SQL SELECT  RESUME
             INTO :clRESUME
             FROM  EMP_PHOTO_RESUME
            WHERE  EMPNO = :hvEMPNO;

  if( SQLCODE == 100 )
    {
      status = NOT_OK;
      printf( "*************************************************\n" );
      printf( "*** ERROR: DSN8DLRV DB2 Sample Program\n"           );
      printf( "***        No entry in the Employee Photo/Resume\n"  );
      printf( "***        table for employee with empno = %s\n",
                          hvEMPNO                                  );
      printf( "***        Processing terminated\n"                 );
      printf( "*************************************************\n" );
    }
  else if( SQLCODE == -305 )
    {
      status = NOT_OK;
      printf( "*************************************************\n" );
      printf( "*** ERROR: DSN8DLRV DB2 Sample Program\n"           );
      printf( "***        No resume data exists in the\n"          );
      printf( "***        Employee Photo/Resume table for the\n"    );
      printf( "***        employee with empno = %s\n",
                          hvEMPNO                                  );
      printf( "***        Processing terminated\n"                 );
      printf( "*************************************************\n" );
    }
  else if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "getEmplResume @ SELECT" );
    }

} /* end getEmplResume */
```

```
void formatEmplResume( void )
/*********************************************************************
* Called by the main routine.  Calls routines to parse out the       *
* contents of the resume into ISPF-shared variables.                 *
*********************************************************************/
{
  /*******************************************************************
  * Get the employee's name, address, and other personal information *
  *******************************************************************/
  getPersonalData();

  /*******************************************************************
  * Get the employee's department no., manager, and other dept data  *
  *******************************************************************/
  if( status == OK )
    getDepartmentData();

  /*******************************************************************
  * Get the employee's education data                                *
  *******************************************************************/
  if( status == OK )
    getEducationData();

  /*******************************************************************
  * Get the employee's employment history                            *
  *******************************************************************/
  if( status == OK )
    getWorkHistoryData();

  /*******************************************************************
  * Free the CLOB locator for the resume                             *
  *******************************************************************/
  if( status == OK )
    {
      EXEC SQL FREE LOCATOR :clRESUME;
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "formatEmplResume @ FREE LOCATOR" );
        }
    }

} /* end formatEmplResume */


void getPersonalData( void )
/*********************************************************************
* Called by the formatEmplResume routine to parse the CLOB locator   *
* data for the employee's name, address, home telephone no., birth-  *
* date, sex, marital status, height, and weight into ISPF variables. *
*********************************************************************/
{
  /*******************************************************************
  * Extract the Personal Data section from the CLOB locator          *
  *******************************************************************/
  EXEC SQL SET :begSection            /* locate start of pers. data */
           = POSSTR( :clRESUME, '    Resume:  ' );
  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "getPersonalData @ POSSTR 1" );
    }
  if( status == OK )
    {
      EXEC SQL SET :endSection        /* locate start of dept. data */
           = POSSTR( :clRESUME, '    Department Information   ' );
      if( SQLCODE != 0 )
        {
          status = NOT_OK;
          sql_error( "getPersonalData @ POSSTR 2" );
        }
    }
  if( status == OK )
    {
      EXEC SQL SET :hvRESUME          /* extract what's in between  */
           = SUBSTR( :clRESUME, :begSection, :endSection-:begSection );
      if( SQLCODE == 0 )
        hvRESUME.data[hvRESUME.length] = '\0';
      else
        {
          status = NOT_OK;
```

```
         sql_error( "getPersonalData @ SUBSTR" );
       }
    }
  /******************************************************************
  * Get the employee's name                                         *
  ******************************************************************/
  if( status == OK )
    {
      phvRESUME = &hvRESUME.data[0];  /* set pointer to the data    */
      begField                        /* find Resume:  label        */
         = strstr( phvRESUME,"  Resume: " );
      begField = begField + 11;       /* skip past label            */
      endField                        /* find Personal Inf... label */
         = strstr( phvRESUME,"    Personal Information   " );
      strncpy( D8EMNAME,              /* get name from in between   */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's street address                               *
  ******************************************************************/
  if( status == OK )
    {
      begField                        /* find Address:  label       */
         = strstr( phvRESUME,"  Address:            " );
      begField = begField + 22;       /* skip past label            */
      endField                        /* find end of street addr    */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMADR1,              /* get addr from in between   */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's city, state, and zipcode                     *
  ******************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to city/st/zip dat */
      endField                        /* find end of ciy/st/zip     */
         = strstr( phvRESUME,"  Phone:              " );
      strncpy( D8EMADR2,              /* get data from in between   */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's home telephone number                        *
  ******************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to home phone data */
      endField                        /* find end of home phone no. */
         = strstr( phvRESUME,"  Birthdate:          " );
      strncpy( D8EMADR3,              /* get phone# from in between */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's birthdate                                    *
  ******************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to birthdate data  */
      endField                        /* find end of birthdate data */
         = strstr( phvRESUME,"  Sex:                " );
      strncpy( D8EMBORN,              /* get birthdate from in betw */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's sex                                          *
  ******************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to sex data        */
      endField                        /* find end of sex data       */
         = strstr( phvRESUME,"  Marital Status:     " );
      strncpy( D8EMSEX,               /* get sex data from in betw  */
               begField,
               endField - begField );
    }
  /******************************************************************
  * Get the employee's marital status                               *
```

```
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to marital status  */
      endField                        /* find end of marital stat.  */
         = strstr( phvRESUME,"  Height:             " );
      strncpy( D8EMPMST,              /* get mar stat from in betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's height                                      *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to height data     */
      endField                        /* find end of height data    */
         = strstr( phvRESUME,"  Weight:             " );
      strncpy( D8EMHGT,               /* get height from in between */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's weight                                      *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to weight data     */
      strcpy( D8EMWGT,                /* weight is at end of string */
               begField );
    }
} /* end getPersonalData */


void getDepartmentData( void )
/*******************************************************************
* Called by the formatEmplResume routine to parse the CLOB locator   *
* data for the employee's department number, manager, job position,  *
* work telephone no., and hire date into ISPF variables.            *
*******************************************************************/
{
  /*****************************************************************
  * Extract the Department Data section from the CLOB locator       *
  *****************************************************************/
  begSection = endSection;            /* Locate start of Dept data  */
  EXEC SQL SET :endSection            /* Locate start of Educ data  */
           = POSSTR( :clRESUME, '    Education   ' );
  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "getDepartmentData @ POSSTR" );
    }
  if( status == OK )
    {
      EXEC SQL SET :hvRESUME          /* extract what's in between  */
           = SUBSTR( :clRESUME, :begSection, :endSection-:begSection );
      if( SQLCODE == 0 )
        hvRESUME.data[hvRESUME.length] = '\0';
      else
        {
          status = NOT_OK;
          sql_error( "getDepartmentData @ SUBSTR" );
        }
    }
  /*****************************************************************
  * Get the employee's department number                           *
  *****************************************************************/
  if( status == OK )
    {
      phvRESUME = &hvRESUME.data[0];  /* set pointer to the data    */
      begField                        /* find Dept Number: label    */
         = strstr( phvRESUME,"  Dept Number:        " );
      begField = begField + 22;       /* skip past label            */
      endField                        /* find end of dept. no.      */
         = strstr( phvRESUME,"  Manager:            " );
      strncpy( D8EMDEPT,              /* get dept# from in between  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's manager's name                              *
  *****************************************************************/
```

```
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to manager data    */
      endField                        /* find end of manager        */
         = strstr( phvRESUME,"  Position:           " );
      strncpy( D8MGRNAM,              /* get mgr name from in betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's job position                                *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to position data   */
      phvRESUME = begField;           /* skip ahead in buffer       */
      endField                        /* find end of position data  */
         = strstr( phvRESUME,"  Phone:              " );
      strncpy( D8EMPOSN,              /* get position from in betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's work telephone number                       *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to work phone data */
      endField                        /* find end of work phone no. */
         = strstr( phvRESUME,"  Hire Date:          " );
      strncpy( D8EMPHON,              /* get work ph# from in betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get the employee's hire date                                   *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to hire date data  */
      strcpy( D8EMHIRE,               /* hire data is at end of str */
               begField );
    }
} /* end getDepartmentData */


void getEducationData( void )
/*******************************************************************
* Called by the formatEmplResume routine to parse the CLOB locator *
* data for the employee's degree dates, descriptions, and schools  *
* into ISPF variables.                                             *
*******************************************************************/
{
  /*****************************************************************
  * Extract the Education Data section from the CLOB locator       *
  *****************************************************************/
  begSection = endSection;            /* Locate start of Educ data  */
  EXEC SQL SET :endSection            /* Locate start of Work Hist  */
          = POSSTR( :clRESUME, '    Work History   ' );
  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "getEducationData @ POSSTR" );
    }
  if( status == OK )
    {
      EXEC SQL SET :hvRESUME          /* extract what's in between  */
          = SUBSTR( :clRESUME, :begSection, :endSection-:begSection );
      if( SQLCODE == 0 )
        hvRESUME.data[hvRESUME.length] = '\0';
      else
        {
          status = NOT_OK;
          sql_error( "getEducationData @ SUBSTR" );
        }
    }
  /*****************************************************************
  * Get year and description of employee's most recent degree      *
  *****************************************************************/
  if( status == OK )
    {
      phvRESUME = &hvRESUME.data[0];  /* set pointer to the data    */
```

```
      begField                        /* find Education label       */
         = strstr( phvRESUME,"    Education   " );
      begField = begField + 16;       /* skip past label            */
      endField                        /* find end of dept. no.      */
         = strstr( phvRESUME,"                " );
      strncpy( D8EMEDY1,              /* get dept# from in between  */
               begField,
               endField - begField );
      begField = endField + 16;       /* set loc to degree descript */
      endField                        /* find end of deg descr data */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMEDD1,              /* get deg descr from in betw */
               begField,
               endField - begField );
    }
  /****************************************************************
  * Get institution that granted employee's most recent degree      *
  ****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to inst name data  */
      phvRESUME = begField;           /* point to beginning         */
      endField                        /* find end of inst name data */
         = strstr( phvRESUME,"   " );
      strncpy( D8EMEDI1,              /* get inst name from in betw */
               begField,
               endField - begField );
    }
  /****************************************************************
  * Get year and description of employee's previous degree          *
  ****************************************************************/
  if( status == OK )
    {
      begField = endField + 3;        /* set loc to grad year data  */
      endField                        /* find end of grad year data */
         = strstr( phvRESUME,"                " );
      strncpy( D8EMEDY2,              /* get hire data from in betw */
               begField,
               endField - begField );
      begField = endField + 16;       /* set loc to degree descript */
      endField                        /* find end of deg descr data */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMEDD2,              /* get deg descr from in betw */
               begField,
               endField - begField );
    }
  /****************************************************************
  * Get institution that granted employee's previous degree         *
  ****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to inst name data  */
      phvRESUME = begField;           /* reset starting point       */
      strcpy( D8EMEDI2,               /* inst name is at end of str */
               begField );
    }
} /* end getEducationData */


void getWorkHistoryData( void )
/*****************************************************************
* Called by the formatEmplResume routine to parse the CLOB locator   *
* data for the employee's job dates, titles, and descriptions into   *
* ISPF variables.                                                    *
*****************************************************************/
{
  /****************************************************************
  * Extract the Work History Data section from the CLOB locator     *
  ****************************************************************/
  begSection = endSection;            /* Locate start of Work Hist  */
  EXEC SQL SET :endSection            /* Locate start of Interests  */
           = POSSTR( :clRESUME, '    Interests   ' );
  if( SQLCODE != 0 )
    {
      status = NOT_OK;
      sql_error( "getWorkHistoryData @ POSSTR" );
    }
  if( status == OK )
    {
      EXEC SQL SET :hvRESUME          /* extract what's in between  */
           = SUBSTR( :clRESUME, :begSection, :endSection-:begSection );
      if( SQLCODE == 0 )
```

```
        hvRESUME.data[hvRESUME.length] = '\0';
      else
        {
          status = NOT_OK;
          sql_error( "getWorkHistoryData @ SUBSTR" );
        }
    }
  /*****************************************************************
  * Get dates and title of employee's most recent job              *
  *****************************************************************/
  if( status == OK )
    {
      phvRESUME = &hvRESUME.data[0];  /* set pointer to the data    */
      begField                        /* find Work History label    */
         = strstr( phvRESUME,"    Work History   " );
      begField = begField + 19;       /* set loc to job 1 dates     */
      phvRESUME = begField;           /* reset starting point       */
      strncpy( D8EMWHD1,              /* job 1 dates, next 15 bytes */
               begField,
               15 );
      begField = begField + 20;       /* set loc to job 1 title     */
      endField                        /* find end of job 1 title    */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMWHT1,              /* get job 1 title from betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get description of employee's most recent job                  *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to job 1 descr.    */
      phvRESUME = begField;           /* reset starting point       */
      endField                        /* find end of job 1 descr.   */
         = strstr( phvRESUME,".   " );
      if( endField - begField < 62 )  /* job 1 descr has 1 part     */
          strncpy( D8EMWHJ1,          /* get job 1 descr from betw  */
                   begField,
                   endField - begField );
      else                            /* job 1 descr has 2 parts    */
        {
          endField                    /* find 1st part of job descr */
             = strstr( phvRESUME,"                      " );
          strncpy( D8EMWHJ1,          /* get job 1 descr from betw  */
                   begField,
                   endField - begField );
          begField = endField + 22;   /* set loc to 2nd part job des*/
          endField                    /* find end of job 1 descr.   */
             = strstr( phvRESUME,".   " );
          strncat( D8EMWHJ1,          /* get rest of job 1 descr.   */
                   begField-1,
                   endField - (begField-1) );
        }
    }
  /*****************************************************************
  * Get dates and title of employee's previous job                 *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 4;        /* set loc to job 2 dates     */
      phvRESUME = begField;           /* reset starting point       */
      strncpy( D8EMWHD2,              /* job 2 dates, next 15 bytes */
               begField,
               15 );
      begField = begField + 20;       /* set loc to job 2 title     */
      endField                        /* find end of job 2 title    */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMWHT2,              /* get job 2 title from betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get description of employee's previous job                     *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to job 2 descr.    */
      phvRESUME = begField;           /* reset starting point       */
      endField                        /* find end of job 2 descr.   */
         = strstr( phvRESUME,".   " );
      if( endField - begField < 62 )  /* job 2 descr has 1 part     */
```

```
          strncpy( D8EMWHJ2,          /* get job 2 title from betw  */
                   begField,
                   endField - begField );
      else                            /* job 2 descr has 2 parts    */
        {
          endField                    /* find 1st part of job descr */
             = strstr( phvRESUME,"                      " );
          strncpy( D8EMWHJ2,          /* get job 2 descr from betw  */
                   begField,
                   endField - begField );
          begField = endField + 22;   /* set loc to 2nd part job des*/
          endField                    /* find end of job 2 descr.   */
             = strstr( phvRESUME,".   " );
          strncat( D8EMWHJ2,          /* get rest of job 2 descr.   */
                   begField-1,
                   endField - (begField-1) );
        }
    }
  /*****************************************************************
  * Get dates and title of employee's other previous job          *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 4;        /* set loc to job 3 dates     */
      phvRESUME = begField;           /* reset starting point       */
      strncpy( D8EMWHD3,              /* job 3 dates, next 15 bytes */
               begField,
               15 );
      begField = begField + 20;       /* set loc to job 3 title     */
      endField                        /* find end of job 3 title    */
         = strstr( phvRESUME,"                      " );
      strncpy( D8EMWHT3,              /* get job 3 title from betw  */
               begField,
               endField - begField );
    }
  /*****************************************************************
  * Get description of employee's other previous job              *
  *****************************************************************/
  if( status == OK )
    {
      begField = endField + 22;       /* set loc to job 3 descr.    */
      phvRESUME = begField;           /* reset starting point       */
      begField = phvRESUME;           /* reset starting point       */
      endField                        /* find end of job 3 descr.   */
         = strstr( phvRESUME,"." );
      if( endField - begField < 62 )  /* job 3 descr has 1 part     */
          strncpy( D8EMWHJ3,          /* get job 3 title from betw  */
                   begField,
                   endField - begField );
      else                            /* job 3 descr has 2 parts    */
        {
          endField                    /* find 1st part of job descr */
             = strstr( phvRESUME,"                      " );
          strncpy( D8EMWHJ3,          /* get job 3 descr from betw  */
                   begField,
                   endField - begField );
          begField = endField + 22;   /* set loc to 2nd part job des*/
          endField                    /* find end of job 3 descr.   */
             = strstr( phvRESUME,".   " );
          strncat( D8EMWHJ3,          /* get rest of job 3 descr.   */
                   begField-1,
                   endField - (begField-1) );
        }
    }
} /* end getWorkHistoryData */


void showEmplResume( void )
/*******************************************************************
* Called by the main routine.  Displays an ISPF panel that is for-   *
* matted with the resume data for the employee specified.           *
*******************************************************************/
{

  ISPFrc = isplink( "DISPLAY ","DSN8SSR " );

} /* end showEmplResume */


void freeISPFvars( void )
/*******************************************************************
* Called by the main routine.  Frees the ISPF variables that were    *
```

```
* established for running this application.                        *
*******************************************************************/
{

  ISPFrc = isplink( VRESET );

} /* end freeISPFvars */


void sql_error( char *locmsg )
/*******************************************************************
* SQL error handler                                                *
*******************************************************************/
{
  short int   rc;                     /* DSNTIAR Return code       */
  int         j,k;                    /* Loop control              */
  static int  lrecl = TIAR_LEN;       /* Width of message lines    */

  /*****************************************************************
  * print the location message                                     *
  *****************************************************************/
  printf( "*****************************************************\n" );
  printf( "*** ERROR: DSN8DLRV DB2 Sample Program\n"                 );
  printf( "***        Unexpected SQLCODE encountered at location\n" );
  printf( "***         %.68s\n", locmsg                              );
  printf( "***        Error detailed below\n"                        );
  printf( "***        Processing terminated\n"                       );
  printf( "*****************************************************\n" );

  /*****************************************************************
  * format and print the SQL message                               *
  *****************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<TIAR_DIM; j++ )
      {
        for( k=0; k<TIAR_LEN; k++ )
          putchar(error_message.error_text[j][k] );
        putchar('\n');
      }
  else
    {
      printf( " *** ERROR: DSNTIAR could not format the message\n" );
      printf( " ***        SQLCODE is %d\n",SQLCODE );
      printf( " ***        SQLERRM is \n" );
      for( j=0; j<sqlca.sqlerrml; j++ )
        printf( "%c", sqlca.sqlerrmc[j] );
      printf( "\n" );
    }

} /* end sql_error */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSN8DLPV

Prompts the user to choose an employee, then retrieves the PSEG photo image for that employee from the PSEG_- PHOTO column of the EMP_PHOTO_RESUME table and passes it to GDDM for formatting and display.

```
/*******************************************************************
* Module name = DSN8DLPV (DB2 sample program)                      *
*                                                                  *
* DESCRIPTIVE NAME = Display PSEG photo image of a specified employee*
*                                                                  *
*                                                                  *
*  LICENSED MATERIALS - PROPERTY OF IBM                            *
*  5655-DB2                                                        *
*  (C) COPYRIGHT 1997 IBM CORP.  ALL RIGHTS RESERVED.              *
*                                                                  *
*  STATUS = VERSION 6                                              *
*                                                                  *
* Function: Prompts the user to choose an employee, then retrieves *
*           the PSEG photo image for that employee from the PSEG_- *
*           PHOTO column of the EMP_PHOTO_RESUME table and passes it *
```

```
*          to GDDM for formatting and display.                         *
*                                                                     *
* Notes:                                                              *
*   Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher       *
*                 Requires IBM Graphical Data Display Manager (GDDM) *
*                 V3R1 or higher                                     *
*                                                                     *
*   Restrictions:                                                     *
*                                                                     *
* Module type: C program                                              *
*   Processor: IBM C/C++ for OS/390 V1R3 or subsequent release       *
* Module size: See linkedit output                                    *
*  Attributes: Re-entrant and re-usable                               *
*                                                                     *
* Entry Point: CEESTART (Language Environment entry point)            *
*     Purpose: See Function                                           *
*     Linkage: Standard MVS program invocation, no parameters         *
*                                                                     *
* Normal Exit: Return Code = 0000                                     *
*              - Message: none                                        *
*                                                                     *
*  Error Exit: Return Code = 0008                                     *
*              - Message: *** ERROR: DSN8DLPV DB2 Sample Program     *
*                                    Unexpected SQLCODE encountered  *
*                                    at location xxx                 *
*                                    Error detailed below            *
*                                    Processing terminated           *
*                                    (DSNTIAR-formatted message here)*
*                                                                     *
*              - Message: *** ERROR: DSN8DLPV DB2 Sample Program     *
*                                    No entry in the Employee Photo/ *
*                                    Resume table for employee with  *
*                                    empno = xxxxxx                  *
*                                    Processing terminated           *
*                                                                     *
*              - Message: *** ERROR: DSN8DLPV DB2 Sample Program     *
*                                    No PSEG photo image exists in   *
*                                    the Employee Photo/Resume table *
*                                    for the employee with empno =   *
*                                    xxxxxx.                         *
*                                    Processing terminated           *
*                                                                     *
*    External References:                                             *
*             - Routines/Services: DSNTIAR, GDDM, ISPF                *
*             - Data areas       : DSNTIAR error_message              *
*             - Control blocks   : None                               *
*                                                                     *
*                                                                     *
*  Pseudocode:                                                        *
*   DSN8DLPV:                                                         *
*   - Do until the user indicates termination                         *
*     - Call getEmplNum to request an employee id                     *
*     - Call getEmplPhoto to retrieve the PSEG photo image            *
*     - Call showEmplPhoto to display the photo                       *
*   End DSN8DLRV                                                      *
*                                                                     *
*   getEmplNum:                                                       *
*   - prompt user to select an employee whose photo is to be viewed  *
*                                                                     *
*   getEmplPhoto:                                                     *
*   - Fetch the specified employee's PSEG photo image from DB2       *
*     - call sql_error for unexpected SQLCODEs                        *
*   End getEmplPhoto:                                                 *
*                                                                     *
*   showEmplPhoto:                                                    *
*   - Use GDDM calls to format and display the PSEG photo image      *
*                                                                     *
*   sql_error:                                                        *
*   - call DSNTIAR to format the unexpected SQLCODE.                  *
*                                                                     *
*********************************************************************/

/******************* C Program Product Libraries ********************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


/******* GDDM Program Product Libraries (Reentrant Versions) ********/
#include <ADMUCIRA>
#include <ADMTSTRC>
#include <ADMUCIRF>
```

```
#include <ADMUCIRG>
#include <ADMUCIRI>


/***************************** Equates ******************************/
#define     NO                  0     /* Boolean: False             */
#define     YES                 1     /* Boolean: True              */

#define     NOT_OK              0     /* Run status indicator: Error*/
#define     OK                  1     /* Run status indicator: Good */

#define     TIAR_DIM           10     /* Max no. of DSNTIAR msgs    */
#define     TIAR_LEN           80     /* Length of DSNTIAR messages */


/************************** Global Storage **************************/
int         keepViewing     = YES;    /*                            */
int         status          = OK;     /* run status                 */

short int   ISPFrc;                   /* For ISPF return code       */


/******************** DB2 SQL Communication Area ********************/
EXEC SQL INCLUDE SQLCA;


/********************** DB2 Message Formatter ***********************/
struct      error_struct {            /* DSNTIAR message structure  */
  short int   error_len;
  char        error_text[TIAR_DIM][TIAR_LEN];
  }         error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar( struct      sqlca         *sqlca,
                          struct      error_struct  *msg,
                          int                       *len );


/**************************** DB2 Tables ****************************/
EXEC SQL DECLARE  EMP_PHOTO_RESUME  TABLE
  (        EMPNO  CHAR(06)     NOT NULL,
       EMP_ROWID  ROWID,
      PSEG_PHOTO  BLOB( 500K ),
       BMP_PHOTO  BLOB( 100K ),
          RESUME  CLOB(   5K )         );


/************** DB2 Host and Null Indicator Variables ***************/
EXEC SQL BEGIN DECLARE SECTION;
  char        hvEMPNO[7];             /*                            */

  SQL TYPE IS BLOB(500K) hvPSEG_PHOTO;/*                            */
  short int   niPSEG_PHOTO = 0;       /*                            */
EXEC SQL END DECLARE SECTION;


/************************** GDDM Variables **************************/
union{ Admaab      AABtag;
       char        AABstr[8];
     } AAB;
int         appl_id;                  /* id for application image   */

                                      /* appl. image vars follow    */
int         ih_pixels       = 800;    /* -horiz size in # of pixels */
int         iv_pixels       = 750;    /* -vert size in # of pixels  */
int         iim_type        =   1;    /* -pixel type   (1=bi-level) */
int         ires_type       =   1;    /* -defined resolution indic. */
int         ires_unit       =   0;    /* -resolut'n units (0=inches)*/
float       ih_res          = 100.00; /* -horizontal resolution     */
float       iv_res          = 100.00; /* -vertical resolution       */

                                      /* PSEG/GDDM convers'n factors*/
int         PSEGformat      =  -3;    /* indicates PSEG format      */
int         PSEGcompression =   4;    /* indicates IBM 3800 compresn*/

int         attype;                   /* type of attn/interrupt key */
int         attval;                   /* value of attn/interrupt key*/
int         count;                    /* number of fields modified  */


/*************************** ISPF Linkage ***************************/
```

```
#pragma linkage(isplink,OS)


/*************************** ISPF Syntax ****************************/
char        CHAR[9]      = "CHAR    ";
char        CONTROL[9]   = "CONTROL ";
char        DISPLAY[9]   = "DISPLAY ";
char        LINE[9]      = "LINE    ";
char        VDEFINE[9]   = "VDEFINE ";
char        VGET[9]      = "VGET    ";
char        VRESET[9]    = "VRESET  ";


/********************** ISPF Shared Variables ***********************/
char        D8EMNUMB[7];             /*                              */


/************************* Global Functions *************************/
int  main( void );                   /*                              */
void getEmplNum( void );             /*                              */
void getEmplPhoto( void );           /*                              */
void showEmplPhoto( void );          /*                              */
void sql_error( char *locmsg );      /*                              */


/*******************************************************************/
/*************************** main routine ***************************/
/*******************************************************************/
int  main( void )
{
  /******************************************************************
  * Display employee photos until user indicates completion          *
  ******************************************************************/
  keepViewing = YES;
  while( keepViewing == YES )
    {
      /**************************************************************
      * prompt user to select an employee whose photo is to be viewed*
      **************************************************************/
      getEmplNum();

      if( keepViewing == YES  &&  status == OK )
        {
          /**********************************************************
          * extract the employee's PSEG photo image from BLOB storage*
          **********************************************************/
          getEmplPhoto();
          /**********************************************************
          * if okay, convert PSEG image to GDDM format and display it*
          **********************************************************/
          if( status == OK )
             showEmplPhoto();
          /**********************************************************
          * otherwise, exit this program                             *
          **********************************************************/
          else
            keepViewing = NO;
        }
    }

} /* end main */


void getEmplNum( void )
/*******************************************************************
* Called by the main routine.  Displays an ISPF panels to prompt the *
* user to select an employee whose photo image is to be displayed.   *
*******************************************************************/
{
  /******************************************************************
  * Share the ISPF var having the employee number                    *
  ******************************************************************/
  ISPFrc = isplink( VDEFINE, "D8EMNUMB", D8EMNUMB, CHAR, 6 );
  strcpy( D8EMNUMB,"      " );

  /******************************************************************
  * Display the prompt panel                                         *
  ******************************************************************/
  ISPFrc = isplink( DISPLAY,"DSN8SSE " );
  if( ISPFrc != 0 )
    keepViewing = NO;
```

```
  /******************************************************************
  * Save off the value of the ISPF shared variable                 *
  ******************************************************************/
  strcpy( hvEMPNO,D8EMNUMB );

  /******************************************************************
  * And release it                                                  *
  ******************************************************************/
  ISPFrc = isplink( VRESET );
} /* end getEmplNum */


void getEmplPhoto( void )
/*********************************************************************
* Called by the main routine. Extracts a specified employee's PSEG   *
* photo image from a BLOB column in the sample EMP_PHOTO_RESUME.     *
* This image will be converted to GDDM format and displayed by the   *
* rotuien showEmplPhoto.                                              *
*********************************************************************/
{
  EXEC SQL SELECT  PSEG_PHOTO
             INTO :hvPSEG_PHOTO
             FROM  EMP_PHOTO_RESUME
            WHERE  EMPNO = :hvEMPNO;


  if( SQLCODE == 0 )
    hvPSEG_PHOTO.data[hvPSEG_PHOTO.length] = '\n';
  else if( SQLCODE == 100 )
    {
      status = NOT_OK;
      printf( "**************************************************\n" );
      printf( "*** ERROR: DSN8DLPV DB2 Sample Program\n"            );
      printf( "***        No entry in the Employee Photo/Resume\n"  );
      printf( "***        table for employee with empno = %s\n",
                          hvEMPNO                                    );
      printf( "***        Processing terminated\n"                  );
      printf( "**************************************************\n" );
    }
  else if( SQLCODE == -305 )
    {
      status = NOT_OK;
      printf( "**************************************************\n" );
      printf( "*** ERROR: DSN8DLPV DB2 Sample Program\n"            );
      printf( "***        No PSEG photo image exists in the\n"      );
      printf( "***        Employee Photo/Resume table for the\n"    );
      printf( "***        employee with empno = %s\n",
                          hvEMPNO                                    );
      printf( "***        Processing terminated\n"                  );
      printf( "**************************************************\n" );
    }
  else
    {
      status = NOT_OK;
      sql_error( "getEmplPhoto @1" );
    }

} /* end getEmplPhoto */


void showEmplPhoto( void )
/*********************************************************************
* Called by the main routine.  Converts the employee's photo from    *
* PSEG format to a GDDM image and then displays it until the user    *
* depresses any PF key or the <enter> key.                           *
*********************************************************************/
{
  /******************************************************************
  * Signal ISPF to full-screen refresh when GDDM session terminates *
  ******************************************************************/
  isplink( CONTROL,DISPLAY,LINE );

  /******************************************************************
  * Initialize GDDM                                                 *
  ******************************************************************/
  fsinit( AAB.AABstr );              /* GDDM anchor block          */

  /******************************************************************
  * Obtain a GDDM application image id                              *
  ******************************************************************/
  imagid( AAB.AABstr,                /* GDDM anchor block          */
          &appl_id );                /* application id for image   */
```

```
   /******************************************************************
   * Create a GDDM application image to receive the employee photo    *
   ******************************************************************/
   imacrt( AAB.AABstr,                /* GDDM anchor block          */
           appl_id,                   /* target: application image  */
           ih_pixels,                 /* horiz size in # of pixels  */
           iv_pixels,                 /* vert size in # of pixels   */
           iim_type,                  /* pixel type   (1=bi-level)  */
           ires_type,                 /* defined resolution indic.  */
           ires_unit,                 /* resolut'n units (0=inches) */
           ih_res,                    /* horizontal resolution      */
           iv_res );                  /* vertical resolution        */

   /******************************************************************
   * Set up conversion of photo from PSEG format to GDDM format       *
   ******************************************************************/
   imapts( AAB.AABstr,                /* GDDM anchor block          */
           appl_id,                   /* target: application image  */
           0,                         /* GDDM proj. id (0=identity) */
           PSEGformat,                /* source format (PSEG)       */
           PSEGcompression );         /* source compression (3800)  */

   /******************************************************************
   * Perform conversion                                               *
   ******************************************************************/
   imapt( AAB.AABstr,                 /* GDDM anchor block          */
          appl_id,                    /* target: application image  */
          hvPSEG_PHOTO.length,        /* source length              */
          hvPSEG_PHOTO.data );        /* source: employee PSEG photo*/

   /******************************************************************
   * Terminate conversion                                             *
   ******************************************************************/
   imapte( AAB.AABstr,                /* GDDM anchor block          */
           appl_id );                 /* target: application image  */

   /******************************************************************
   * Transfer the GDDM application image to the display               *
   ******************************************************************/
   imxfer( AAB.AABstr,                /* GDDM anchor block          */
           appl_id,                   /* source: application image  */
           0,                         /* target: 0=display          */
           0 );                       /* GDDM proj. id (0=identity) */

   /******************************************************************
   * Disable user updates to the image on the display                 *
   ******************************************************************/
   fsenab( AAB.AABstr,                /* GDDM anchor block          */
           1,                         /* type of input (1=alphanum) */
           0 );                       /* type of control (0=disable)*/

   fsenab( AAB.AABstr,                /* GDDM anchor block          */
           2,                         /* type of input (2=graphic)  */
           0 );                       /* type of control (0=disable)*/

   fsenab( AAB.AABstr,                /* GDDM anchor block          */
           3,                         /* type of input (3=image)    */
           0 );                       /* type of control (0=disable)*/

   /******************************************************************
   * Display the image until attn or interrupt key depressed          *
   ******************************************************************/
   asread( AAB.AABstr,                /* GDDM anchor block          */
           &attype,                   /* type of attn/interrupt key */
           &attval,                   /* value of attn/interrupt key*/
           &count );                  /* number of fields modified  */

   /******************************************************************
   * Delete the GDDM application image                                *
   ******************************************************************/
   imadel( AAB.AABstr,                /* GDDM anchor block          */
           appl_id );                 /* target: application image  */

   /******************************************************************
   * Terminate GDDM                                                   *
   ******************************************************************/
   fsterm( AAB.AABstr );              /* GDDM anchor block          */

 } /* end showEmplPhoto */
```

```
void sql_error( char *locmsg )
/*******************************************************************
* SQL error handler                                                *
*******************************************************************/
{
  short int   rc;                     /* DSNTIAR Return code        */
  int         j,k;                    /* Loop control               */
  static int  lrecl = TIAR_LEN;       /* Width of message lines     */

  /*****************************************************************
  * print the location message                                     *
  *****************************************************************/
  printf( "*****************************************************\n" );
  printf( "*** ERROR: DSN8DLPV DB2 Sample Program\n"               );
  printf( "***        Unexpected SQLCODE encountered at location\n" );
  printf( "***         %.68s\n", locmsg                             );
  printf( "***        Error detailed below\n"                      );
  printf( "***        Processing terminated\n"                     );
  printf( "*****************************************************\n" );

  /*****************************************************************
  * format and print the SQL message                               *
  *****************************************************************/
  rc = dsntiar( &sqlca, &error_message, &lrecl );
  if( rc == 0 )
    for( j=0; j<TIAR_DIM; j++ )
      {
        for( k=0; k<TIAR_LEN; k++ )
          putchar(error_message.error_text[j][k] );
        putchar('\n');
      }
  else
    {
      printf( " *** ERROR: DSNTIAR could not format the message\n" );
      printf( " ***        SQLCODE is %d\n",SQLCODE );
      printf( " ***        SQLERRM is \n" );
      for( j=0; j<sqlca.sqlerrml; j++ )
        printf( "%c", sqlca.sqlerrmc[j] );
      printf( "\n" );
    }

} /* end sql_error */
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2C

THIS JCL PERFORMS THE PHASE 2 COBOL SETUP FOR THE SAMPLE APPLICATIONS.

```
//********************************************************************* 00000100
//*  NAME = DSNTEJ2C                                                    00000200
//*                                                                     00000300
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00000400
//*                     PHASE 2                                         00000500
//*                     COBOL                                           00000600
//*                                                                     00000700
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00000800
//*    5605-DB2                                                         00000900
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00001000
//*                                                                     00001100
//*    STATUS = VERSION 11                                              00001200
//*                                                                     00001300
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 COBOL SETUP FOR THE       00001400
//*             SAMPLE APPLICATIONS.  IT PREPARES AND EXECUTES          00001500
//*             COBOL BATCH PROGRAMS.                                   00001600
//*                                                                     00001700
//*             THIS JOB IS RUN AFTER PHASE 1.                          00001800
//*                                                                     00001900
//*                                                                     00002000
//*  CHANGE ACTIVITY =                                                  00002100
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00002200
//*               ADD SET CURRENT SQLID                                 00002300
//*                                                                     00002400
//********************************************************************* 00002500
//*                                                                     00002600
//JOBLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT                              00002700
```

```
//        DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//        DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//*        STEP  1: CREATE COPY FILE TABLE DESCRIPTIONS (DCLGEN)
//PH02CS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=F,LRECL=200,BLKSIZE=200)
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDP)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCEM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCAD)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCA2)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCCS)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCOV)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDT)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCED)'
DSN SYSTEM(DSN)
 DCLGEN TABLE(VDEPT)   +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDP)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PDEPT)
 DCLGEN TABLE(VEMP)   +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCEM)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PEMP)
 DCLGEN TABLE(VDEPMG1) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDM)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PDEPMGR)
 DCLGEN TABLE(VASTRDE1) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCAD)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PASTRDET)
 DCLGEN TABLE(VASTRDE2) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCA2)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   NAMES(ADE2)  +
   STRUCTURE(PASTRDE2)
 DCLGEN TABLE(VCONA)  +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCCS)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PCONA)
 DCLGEN TABLE(VOPTVAL) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCOV)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(POPTVAL)
 DCLGEN TABLE(VDSPTXT) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDT)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PDSPTXT)
 DCLGEN TABLE(VEMPDPT1) +
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCED)') +
   ACTION(ADD) APOST +
   LANGUAGE(IBMCOB) +
   STRUCTURE(PEMPDPT1)
END
//*
//*        STEP  2: PREPARE ERROR MESSAGE ROUTINE
//PH02CS02 EXEC DSNHICOB,MEM=DSN8MCG,
//         COND=(4,LT),
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
//         NOXREF,'SQL(DB2)','DEC(31)'),
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
//         PARM.LKED='LIST,XREF,MAP,RENT'
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MCG),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8MCG),
//             DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MCG),
//             DISP=SHR
//*
```

```
//*        STEP  3: PREPARE COBOL PHONE PROGRAM                          00011000
//PH02CS03 EXEC DSNHICOB,MEM=DSN8BC3,                                    00011100
//         COND=(4,LT),                                                  00011200
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,                00011300
//         NOXREF,'SQL(DB2)','DEC(31)'),                                 00011400
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')         00011500
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BC3),                      00011600
//             DISP=SHR                                                  00011700
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                                00011800
//             DISP=SHR                                                  00011900
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8BC3),                          00012000
//             DISP=SHR                                                  00012100
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BC3),                       00012200
//             DISP=SHR                                                  00012300
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                                00012400
//             DISP=SHR                                                  00012500
//LKED.SYSIN   DD *                                                      00012600
     INCLUDE SYSLIB(DSNELI)                                              00012700
     INCLUDE RUNLIB(DSN8MCG)                                             00012800
//*                                                                      00012900
//*        STEP  4: BIND AND RUN PROGRAMS                                00013000
//PH02CS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                     00013100
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                           00013200
//SYSTSPRT DD SYSOUT=*                                                   00013300
//SYSPRINT DD SYSOUT=*                                                   00013400
//SYSUDUMP DD SYSOUT=*                                                   00013500
//SYSOUT   DD SYSOUT=*                                                   00013600
//REPORT   DD SYSOUT=*                                                   00013700
//SYSIN    DD *                                                          00013800
 SET CURRENT SQLID = 'SYSADM';                                           00013900
 GRANT BIND, EXECUTE ON PLAN DSN8BH!!                                    00014000
   TO PUBLIC;                                                            00014100
//SYSTSIN DD *                                                           00014200
 DSN SYSTEM(DSN)                                                         00014300
 BIND PACKAGE(DSN8BH!!) MEMBER(DSN8BC3) +                                00014400
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 00014500
 BIND PLAN(DSN8BH!!) PKLIST(DSN8BH!!.*) +                                00014600
      ACTION(REPLACE) RETAIN +                                           00014700
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          00014800
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                 00014900
      LIB('DSN!!0.RUNLIB.LOAD')                                          00015000
 RUN  PROGRAM(DSN8BC3) PLAN(DSN8BH!!) -                                  00015100
      LIB('DSN!!0.RUNLIB.LOAD')                                          00015200
 END                                                                     00015300
//CARDIN   DD *                                                          00015400
L*                                                                       00015500
LJO%                                                                     00015600
L%SON                                                                    00015700
LSMITH                                                                   00015800
LBROWN          ALAN                                                     00015900
LBROWN          DAVID                                                    00016000
U                           0002304265                                   00016100
//*                                                                      00016200
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2D

THIS JCL PERFORMS THE PHASE 2 C LANGUAGE SETUP FOR THE SAMPLE APPLICATIONS.

```
//********************************************************************** 00010000
//*  NAME = DSNTEJ2D                                                     00020000
//*                                                                      00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                           00040000
//*                     PHASE 2                                          00050000
//*                     C                                                00060000
//*                                                                      00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                              00080000
//*    5605-DB2                                                          00090000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.          00100000
//*                                                                      00110000
//*    STATUS = VERSION 11                                               00120001
//*                                                                      00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 C LANGUAGE SETUP FOR       00140000
//*             THE SAMPLE APPLICATIONS.  IT PREPARES AND EXECUTES       00150000
//*             C BATCH PROGRAMS.                                        00160000
```

```
//*                                                                     00170000
//*     NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF       00180000
//*             THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION         00190000
//*                                                                     00200000
//*             THIS JOB IS RUN AFTER PHASE 1.                          00210000
//*                                                                     00220000
//* CHANGE ACTIVITY =                                                   00230000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00231000
//*               ADD SET CURRENT SQLID                                 00231103
//*                                                                     00232000
//********************************************************************* 00240000
//*                                                                     00250000
//JOBLIB  DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00260000
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                           00270000
//*                                                                     00280000
//*        STEP 1 : PREPARE ERROR MESSAGE ROUTINE                       00290000
//PH02DS01 EXEC DSNHC,MEM=DSN8MDG,                                      00300000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00310000
//             SOURCE,XREF),                                            00320000
//         PARM.C='SOURCE XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',       00330000
//         PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'                      00340000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MDG),                     00350000
//             DISP=SHR                                                 00360000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00370000
//             DISP=SHR                                                 00380000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8MDG),                         00390000
//             DISP=SHR                                                 00400000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MDG),                      00410000
//             DISP=SHR                                                 00420000
//*                                                                     00430000
//*        STEP 2 : PREPARE C PHONE PROGRAM                             00440000
//PH02DS02 EXEC DSNHC,MEM=DSN8BD3,                                      00450000
//         COND=(4,LT),                                                 00460000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00470000
//             SOURCE,XREF),                                            00480000
//         PARM.C='SOURCE LIST MARGINS(1,72) OPTFILE(DD:CCOPTS)',       00490000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP'                           00500000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BD3),                     00510000
//             DISP=SHR                                                 00520000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00530000
//             DISP=SHR                                                 00540000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8BD3),                         00550000
//             DISP=SHR                                                 00560000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BD3),                      00570000
//             DISP=SHR                                                 00580000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               00590000
//             DISP=SHR                                                 00600000
//LKED.SYSIN   DD *                                                     00610000
     INCLUDE SYSLIB(DSNELI)                                             00620000
     INCLUDE RUNLIB(DSN8MDG)                                            00630000
//*                                                                     00640000
//*        STEP 3 : BIND AND RUN PROGRAMS                               00650000
//PH02DS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00660000
//DBRMLIB   DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                        00670000
//SYSTSPRT  DD  SYSOUT=*                                                00680000
//SYSPRINT  DD  SYSOUT=*                                                00690000
//CEEDUMP   DD  SYSOUT=*                                                00700000
//SYSUDUMP  DD  SYSOUT=*                                                00710000
//SYSOUT    DD  SYSOUT=*                                                00720000
//REPORT    DD  SYSOUT=*                                                00730000
//SYSIN     DD  *                                                       00740000
 SET CURRENT SQLID = 'SYSADM';                                          00741002
 GRANT BIND, EXECUTE ON PLAN DSN8BD!!                                   00750004
   TO PUBLIC;                                                           00751004
//SYSTSIN   DD  *                                                       00760000
 DSN SYSTEM(DSN)                                                        00770000
 BIND PACKAGE(DSN8BD!!) MEMBER(DSN8BD3) +                               00780000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00781000
 BIND PLAN(DSN8BD!!) PKLIST(DSN8BD!!.*) +                               00782000
      ACTION(REPLACE) RETAIN +                                          00782100
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         00783000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                00790000
      LIB('DSN!!0.RUNLIB.LOAD')                                         00800000
 RUN  PROGRAM(DSN8BD3) PLAN(DSN8BD!!) -                                 00810000
      LIB('DSN!!0.RUNLIB.LOAD')                                         00820000
 END                                                                    00830000
//CARDIN   DD  *                                                        00840000
L*                                                                      00850000
LJO%                                                                    00860000
L%SON                                                                   00870000
LSMITH                                                                  00880000
LBROWN          ALAN                                                    00890000
```

```
LBROWN          DAVID
U                           0002304265
//*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2E

THIS JCL PERFORMS THE PHASE 2 C++ LANGUAGE SETUP FOR THE SAMPLE APPLICATIONS.

```
//*********************************************************************
//*  NAME = DSNTEJ2E
//*
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//*                     PHASE 2
//*                     C++
//*
//*    Licensed Materials - Property of IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.
//*
//*    STATUS = Version 11
//*
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 C++ LANGUAGE SETUP FOR
//*             THE SAMPLE APPLICATIONS.  IT PREPARES AND EXECUTES
//*             C++ BATCH PROGRAMS.
//*
//*     NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF
//*             THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION
//*
//*             THIS JOB IS RUN AFTER PHASE 1.
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651
//*               Add SET CURRENT SQLID                          164462
//*
//*********************************************************************
//*
//JOBLIB  DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//*
//*        STEP 1 : PREPARE ERROR MESSAGE ROUTINE
//PH02ES01 EXEC DSNHC,MEM=DSN8MDG,
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//             SOURCE,XREF),
//         PARM.C='SOURCE XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
//         PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MDG),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8MDG),
//             DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MDG),
//             DISP=SHR
//*
//*        STEP 2 : PREPARE CLASSES USED BY C++ PHONE PROGRAM
//PH02ES02 EXEC DSNHCPP,MEM=DSN8BECL,COND=(4,LT),
//         PARM.PC=('HOST(CPP),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//             SOURCE,XREF),
//         PARM.CP=('/CXX SOURCE XREF OPTFILE(DD:CCOPTS)',
//             'LANGLVL(EXTENDED)'),
//         PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BE3),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8BECL),
//             DISP=SHR
//CP.USERLIB   DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//CP.SYSLIN    DD DSN=&&LOADSET1,
//             DISP=(MOD,PASS),
//             UNIT=SYSDA,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//PLKED.SYSIN  DD DSN=&&LOADSET1,DISP=(OLD,PASS)
```

```
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BECL),                     00670000
//             DISP=SHR                                                 00680000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               00690000
//             DISP=SHR                                                 00700000
//*                                                                     00710000
//*        STEP 3 : PREPARE C++ PHONE PROGRAM                           00720000
//PH02ES03 EXEC DSNHCPP,MEM=DSN8BE3,                                    00730000
//         COND=(4,LT),                                                 00740000
//         PARM.PC=('HOST(CPP),CCSID(1047),MARGINS(1,72),STDSQL(NO)',   00750000
//             SOURCE,XREF),                                            00760000
//         PARM.CP=('/CXX SOURCE XREF OPTFILE(DD:CCOPTS)',              00770000
//             'LANGLVL(EXTENDED)'),                                    00780000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP'                           00790000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DUMMY),                       00800000
//             DISP=SHR                                                 00810000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00820000
//             DISP=SHR                                                 00830000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8BE3),                         00840000
//             DISP=SHR                                                 00850000
//CP.USERLIB   DD DSN=DSN!!0.SDSNSAMP,                                  00860000
//             DISP=SHR                                                 00870000
//PLKED.SYSIN  DD DSN=&&LOADSET,DISP=(OLD,DELETE)                       00880000
//             DD DSN=&&LOADSET1,DISP=(OLD,DELETE)                      00890000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BE3),                      00900000
//             DISP=SHR                                                 00910000
//LKED.SYSIN   DD *                                                     00920000
     INCLUDE SYSLIB(DSNELI)                                             00930000
     INCLUDE SYSLMOD(DSN8MDG)                                           00940000
     NAME DSN8BE3(R)                                                    00950000
//*                                                                     00960000
//*        STEP 4 : BIND AND RUN PROGRAMS                               00970000
//PH02ES04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00980000
//DBRMLIB   DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                        00990000
//SYSTSPRT  DD  SYSOUT=*                                                01000000
//SYSPRINT  DD  SYSOUT=*                                                01010000
//CEEDUMP   DD  SYSOUT=*                                                01020000
//SYSUDUMP  DD  SYSOUT=*                                                01030000
//SYSOUT    DD  SYSOUT=*                                                01040000
//REPORT    DD  SYSOUT=*                                                01050000
//SYSIN     DD  *                                                       01060000
 SET CURRENT SQLID = 'SYSADM';                                          01070000
 GRANT BIND, EXECUTE ON PLAN DSN8BE!!                                   01080000
   TO PUBLIC;                                                           01090000
//SYSTSIN   DD  *                                                       01100000
 DSN SYSTEM(DSN)                                                        01110000
 BIND PACKAGE(DSN8BE!!) MEMBER(DSN8BE3) +                               01120000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01130000
 BIND PLAN(DSN8BE!!) PKLIST(DSN8BE!!.*) +                               01140000
      ACTION(REPLACE) RETAIN +                                          01150000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01160000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                01170000
      LIB('DSN!!0.RUNLIB.LOAD')                                         01180000
 RUN  PROGRAM(DSN8BE3) PLAN(DSN8BE!!) -                                 01190000
      LIB('DSN!!0.RUNLIB.LOAD')                                         01200000
 END                                                                    01210000
//CARDIN   DD  *                                                        01220000
L*                                                                      01230000
LJO%                                                                    01240000
L%SON                                                                   01250000
LSMITH                                                                  01260000
LBROWN          ALAN                                                    01270000
LBROWN          DAVID                                                   01280000
U                           0002304265                                  01290000
//*                                                                     01300000
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSNTEJ2P

THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH PL/I.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ2P                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 2                                         00050000
```

```
//*                     PL/I                                            00060000
//*                                                                     00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00080000
//*    5605-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00100000
//*                                                                     00110000
//*    STATUS = VERSION 11                                              00120000
//*                                                                     00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE      00140000
//*             APPLICATIONS AT SITES WITH PL/I.  IT PREPARES AND       00150000
//*             EXECUTES THE PL/I BATCH PROGRAM.                        00160000
//*                                                                     00170000
//*             THIS JOB IS RUN AFTER PHASE 1.                          00180000
//*                                                                     00190000
//*  CHANGE ACTIVITY =                                                  00200000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00201000
//*               ADD SET CURRENT SQLID                                 00202002
//*                                                                     00210000
//********************************************************************* 00220000
//*                                                                     00230000
//JOBLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT                              00240000
//        DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                              00250000
//        DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN                           00260000
//*        STEP  1: CREATE COPY FILE TABLE DESCRIPTIONS ( DCLGEN )      00270000
//PH02PS01 EXEC PGM=IKJEFT01,DYNAMNBR=20                                00280000
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=F,LRECL=200,BLKSIZE=200)              00290000
//SYSUDUMP DD SYSOUT=*                                                  00300000
//SYSTSIN  DD *                                                         00310000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPAC)'                                   00320000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDP)'                                   00330000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPEM)'                                   00340000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPJ)'                                   00350000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPA)'                                   00360000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPEP)'                                   00370000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDM)'                                   00380000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPAD)'                                   00390000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPA2)'                                   00400000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPR)'                                   00410000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPD)'                                   00420000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPP2)'                                   00430000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPSA)'                                   00440000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPS2)'                                   00450000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPCS)'                                   00460000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPOV)'                                   00470000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDT)'                                   00480000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPED)'                                   00490000
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPFP)'                                   00500000
DSN SYSTEM(DSN)                                                         00510000
 DCLGEN TABLE(VACT) +                                                   00520000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPAC)') +                            00530000
   ACTION(ADD) +                                                        00540000
   LANGUAGE(PLI) +                                                      00550000
   STRUCTURE(PACT)                                                      00560000
 DCLGEN TABLE(VDEPT) +                                                  00570000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDP)') +                            00580000
   ACTION(ADD) +                                                        00590000
   LANGUAGE(PLI) +                                                      00600000
   STRUCTURE(PDEPT)                                                     00610000
 DCLGEN TABLE(VEMP) +                                                   00620000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPEM)') +                            00630000
   ACTION(ADD) +                                                        00640000
   LANGUAGE(PLI) +                                                      00650000
   STRUCTURE(PEMP)                                                      00660000
 DCLGEN TABLE(VPROJ) +                                                  00670000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPJ)') +                            00680000
   ACTION(ADD) +                                                        00690000
   LANGUAGE(PLI) +                                                      00700000
   STRUCTURE(PPROJ)                                                     00710000
 DCLGEN TABLE(VPROJACT) +                                               00720000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPA)') +                            00730000
   ACTION(ADD) +                                                        00740000
   LANGUAGE(PLI) +                                                      00750000
   STRUCTURE(PPROJACT)                                                  00760000
 DCLGEN TABLE(VEMPPROJACT) +                                            00770000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPEP)') +                            00780000
   ACTION(ADD) +                                                        00790000
   LANGUAGE(PLI) +                                                      00800000
   STRUCTURE(PEMPPROJACT)                                               00810000
 DCLGEN TABLE(VDEPMG1) +                                                00820000
   LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDM)') +                            00830000
   ACTION(ADD) +                                                        00840000
   LANGUAGE(PLI) +                                                      00850000
```

```
    STRUCTURE(PDEPMGR)                                                    00860000
  DCLGEN TABLE(VASTRDE1) +                                                00870000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPAD)') +                             00880000
    ACTION(ADD) +                                                         00890000
    LANGUAGE(PLI) +                                                       00900000
    STRUCTURE(PASTRDET)                                                   00910000
  DCLGEN TABLE(VASTRDE2) +                                                00920000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPA2)') +                             00930000
    ACTION(ADD) +                                                         00940000
    LANGUAGE(PLI) +                                                       00950000
    NAMES(ASTD) +                                                         00960000
    STRUCTURE(PASTRDE2)                                                   00970000
  DCLGEN TABLE(VPROJRE1) +                                                00980000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPR)') +                             00990000
    ACTION(ADD) +                                                         01000000
    LANGUAGE(PLI) +                                                       01010000
    STRUCTURE(PPROJRES)                                                   01020000
  DCLGEN TABLE(VPSTRDE1) +                                                01030000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPD)') +                             01040000
    ACTION(ADD) +                                                         01050000
    LANGUAGE(PLI) +                                                       01060000
    STRUCTURE(PPSTRDET)                                                   01070000
  DCLGEN TABLE(VPSTRDE2) +                                                01080000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPP2)') +                             01090000
    ACTION(ADD) +                                                         01100000
    LANGUAGE(PLI) +                                                       01110000
    NAMES(PSTD) +                                                         01120000
    STRUCTURE(PPSTRDE2)                                                   01130000
  DCLGEN TABLE(VSTAFAC1) +                                                01140000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPSA)') +                             01150000
    ACTION(ADD) +                                                         01160000
    LANGUAGE(PLI) +                                                       01170000
    NAMES(STAF) +                                                         01180000
    STRUCTURE(PSTAFAC1)                                                   01190000
  DCLGEN TABLE(VSTAFAC2) +                                                01200000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPS2)') +                             01210000
    ACTION(ADD) +                                                         01220000
    LANGUAGE(PLI) +                                                       01230000
    STRUCTURE(PSTAFACT)                                                   01240000
  DCLGEN TABLE(VCONA) +                                                   01250000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPCS)') +                             01260000
    ACTION(ADD) +                                                         01270000
    LANGUAGE(PLI) +                                                       01280000
    STRUCTURE(PCONA)                                                      01290000
  DCLGEN TABLE(VOPTVAL) +                                                 01300000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPOV)') +                             01310000
    ACTION(ADD) +                                                         01320000
    LANGUAGE(PLI) +                                                       01330000
    STRUCTURE(POPTVAL)                                                    01340000
  DCLGEN TABLE(VDSPTXT) +                                                 01350000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDT)') +                             01360000
    ACTION(ADD) +                                                         01370000
    LANGUAGE(PLI) +                                                       01380000
    STRUCTURE(PDSPTXT)                                                    01390000
  DCLGEN TABLE(VEMPDPT1) +                                                01400000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPED)') +                             01410000
    ACTION(ADD) +                                                         01420000
    LANGUAGE(PLI) +                                                       01430000
    STRUCTURE(PEMPDPT1)                                                   01440000
  DCLGEN TABLE(VFORPLA) +                                                 01450000
    LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPFP)') +                             01460000
    ACTION(ADD) +                                                         01470000
    LANGUAGE(PLI) +                                                       01480000
    STRUCTURE(PFORPLA)                                                    01490000
 END                                                                      01500000
 //*                                                                      01510000
 //*        STEP  2: PREPARE PLI MESSAGE ROUTINE                          01520000
 //PH02PS02 EXEC DSNHPLI,MEM=DSN8MPG,                                     01530000
 //           COND=(4,LT),                                                01540000
 //           PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',       01550000
 //           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',      01560000
 //           'LIMITS(EXTNAME(7)),OPTIONS'),                              01570000
 //           PARM.LKED='NCAL,LIST,XREF'                                  01580000
 //PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8MPG),                          01590000
 //             DISP=SHR                                                  01600000
 //PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MPG),                      01610000
 //             DISP=SHR                                                  01620000
 //PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                                01630000
 //             DISP=SHR                                                  01640000
 //LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MPG),                       01650000
 //             DISP=SHR                                                  01660000
 //*                                                                      01670000
```

```
//*        STEP  3: PREPARE TELEPHONE PROGRAM                           01680000
//PH02PS03 EXEC DSNHPLI,MEM=DSN8BP3,                                    01690000
//           COND=(4,LT),                                               01700000
//           PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',      01710000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            01720000
//           'LIMITS(EXTNAME(7)),OPTIONS')                              01730000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8BP3),                         01740000
//             DISP=SHR                                                 01750000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BP3),                     01760000
//             DISP=SHR                                                 01770000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01780000
//             DISP=SHR                                                 01790000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BP3),                      01800000
//             DISP=SHR                                                 01810000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               01820000
//             DISP=SHR                                                 01830000
//LKED.SYSIN   DD *                                                     01840000
     INCLUDE SYSLIB(DSNELI)                                             01850000
     INCLUDE RUNLIB(DSN8MPG)                                            01860000
//*                                                                     01870000
//*        STEP  4: BIND PROGRAMS                                       01880000
//PH02PS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01890000
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,                                  01900000
//             DISP=SHR                                                 01910000
//SYSTSPRT DD SYSOUT=*                                                  01920000
//SYSPRINT DD SYSOUT=*                                                  01930000
//SYSUDUMP DD SYSOUT=*                                                  01940000
//SYSIN   DD *                                                          01950000
   SET CURRENT SQLID = 'SYSADM';                                        01951001
   GRANT BIND, EXECUTE ON PLAN DSN8BP!!                                 01960003
     TO PUBLIC;                                                         01961003
//SYSTSIN DD *                                                          01970000
 DSN SYSTEM(DSN)                                                        01980000
 BIND PACKAGE(DSN8BP!!) MEMBER(DSN8BP3) +                               01990000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01991000
 BIND PLAN(DSN8BP!!) PKLIST(DSN8BP!!.*) +                               01992000
      ACTION(REPLACE) RETAIN +                                          01992100
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01993000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                 02000000
      LIB('DSN!!0.RUNLIB.LOAD')                                         02010000
 END                                                                    02020000
//*                                                                     02030000
//*        STEP  5: RUN  PROGRAMS                                       02040000
//PH02PS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02050000
//SYSTSPRT DD SYSOUT=*                                                  02060000
//SYSPRINT DD SYSOUT=*                                                  02070000
//REPORT   DD SYSOUT=*                                                  02080000
//SYSUDUMP DD SYSOUT=*                                                  02090000
//CARDIN   DD *                                                         02100000
L*                                                                      02110000
LJO%                                                                    02120000
L%SON                                                                   02130000
LSMITH                                                                  02140000
LBROWN          ALAN                                                    02150000
LBROWN          DAVID                                                   02160000
U                           0002304265                                  02170000
//SYSTSIN DD *                                                          02180000
 DSN SYSTEM(DSN)                                                        02190000
 RUN  PROGRAM(DSN8BP3) PLAN(DSN8BP!!) -                                 02200000
      LIB('DSN!!0.RUNLIB.LOAD')                                         02210000
 END                                                                    02220000
//*                                                                     02230000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2F

THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH FORTRAN.

```
//********************************************************************* 00000100
//*  NAME = DSNTEJ2F                                                    00000200
//*                                                                     00000300
//*  DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                         00000400
//*                     PHASE 2                                         00000500
//*                     FORTRAN                                         00000600
//*                                                                     00000700
```

```
//*    LICENSED MATERIALS - PROPERTY OF IBM                            00000800
//*    5605-DB2                                                        00000900
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.        00001000
//*                                                                    00001100
//*    STATUS = VERSION 11                                             00001200
//*                                                                    00001300
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE     00001400
//*             APPLICATIONS AT SITES WITH FORTRAN.  IT PREPARES       00001500
//*             AND EXECUTES THE FORTRAN BATCH PROGRAM.                00001600
//*                                                                    00001700
//*             THIS JOB IS RUN AFTER PHASE 1.                         00001800
//*                                                                    00001900
//*  CHANGE ACTIVITY =                                                 00002000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00002100
//*               ADD SET CURRENT SQLID                                00002200
//*                                                                    00002300
//********************************************************************* 00002400
//*                                                                    00002500
//JOBLIB   DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                            00002600
//         DD  DSN=SYS1.VSF2FORT,DISP=SHR                              00002700
//*        STEP  1: PREPARE DSNTIR ROUTINE                             00002800
//PH02FS01 EXEC DSNHASM,MEM=DSNTIR,                                    00002900
//         PARM.PC='HOST(ASM),STDSQL(NO)',                             00003000
//         PARM.ASM='RENT,OBJECT,NODECK',                              00003100
//         PARM.LKED='XREF,NCAL'                                       00003200
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSNTIR),                     00003300
//             DISP=SHR                                                00003400
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                              00003500
//             DISP=SHR                                                00003600
//             DD DSN=DSN!!0.SDSNSAMP,                                 00003700
//             DISP=SHR                                                00003800
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSNTIR),                         00003900
//             DISP=SHR                                                00004000
//ASM.SYSLIB   DD DSN=DSN!!0.SDSNSAMP,                                 00004100
//             DISP=SHR                                                00004200
//             DD DSN=SYS1.MACLIB,DISP=SHR                             00004300
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTIR),                      00004400
//             DISP=SHR                                                00004500
//LKED.SYSIN   DD *                                                    00004600
   ENTRY   DSNTIR                                                      00004700
   NAME    DSNTIR(R)                                                   00004800
//*                                                                    00004900
//*        STEP  2: PREPARE TELEPHONE PROGRAM                          00005000
//PH02FS02 EXEC DSNHFOR,MEM=DSN8BF3,                                   00005100
//         COND=(4,LT),                                                00005200
//         PARM.PC='HOST(FORTRAN),SOURCE,XREF,STDSQL(NO)',             00005300
//         PARM.FORT='MAP,GOSTMT,SOURCE,XREF'                          00005400
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BF3),                    00005500
//             DISP=SHR                                                00005600
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                              00005700
//             DISP=SHR                                                00005800
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8BF3),                        00005900
//             DISP=SHR                                                00006000
//LKED.SYSLIB  DD                                                      00006100
//             DD                                                      00006200
//             DD DSN=DSN!!0.RUNLIB.LOAD,                              00006300
//             DISP=SHR                                                00006400
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BF3),                     00006500
//             DISP=SHR                                                00006600
//LKED.SYSIN   DD *                                                    00006700
     INCLUDE SYSLIB(DSNHFT)                                            00006800
//*                                                                    00006900
//*        STEP  3: BIND AND RUN PROGRAM                               00007000
//PH02FS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                   00007100
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                         00007200
//SYSTSPRT DD SYSOUT=*                                                 00007300
//FT06F001 DD SYSOUT=*                                                 00007400
//SYSUDUMP DD SYSOUT=*                                                 00007500
//SYSPRINT DD SYSOUT=*                                                 00007600
//SYSIN    DD *                                                        00007700
   SET CURRENT SQLID = 'SYSADM';                                       00007800
   GRANT BIND, EXECUTE ON PLAN DSN8BF!!                                00007900
     TO PUBLIC;                                                        00008000
//SYSTSIN DD *                                                         00008100
 DSN SYSTEM(DSN)                                                       00008200
 BIND PACKAGE(DSN8BF!!) MEMBER(DSN8BF3) +                              00008300
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)               00008400
 BIND PLAN(DSN8BF!!) PKLIST(DSN8BF!!.*) +                              00008500
      ACTION(REPLACE) RETAIN +                                         00008600
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                        00008700
 RUN  PROGRAM(DSN8BF3) PLAN(DSN8BF!!) -                                00008800
      LIB('DSN!!0.RUNLIB.LOAD')                                        00008900
```

```
  RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                00009000
       LIB('DSN!!0.RUNLIB.LOAD')                                        00009100
  END                                                                   00009200
//FT05F001 DD *                                                         00009300
L*                                                                      00009400
LJO%                                                                    00009500
L%SON                                                                   00009600
LSMITH                                                                  00009700
LBROWN          ALAN                                                    00009800
LBROWN          DAVID                                                   00009900
U                            0002304265                                 00010000
//*                                                                     00010100
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ3C

THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH COBOL.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ3C                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 3                                         00050000
//*                     COBOL, ISPF, CAF                                00060000
//*                                                                     00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00080000
//*    5605-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00100000
//*                                                                     00110000
//*    STATUS = VERSION 11                                              00120000
//*                                                                     00130000
//*    FUNCTION = THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE    00140000
//*               APPLICATIONS AT SITES WITH COBOL.  IT PREPARES THE    00150000
//*               COBOL ISPF CAF TELEPHONE APPLICATION AND THE REMOTE   00160000
//*               COBOL ORGANIZATION APPLICATION.                       00170000
//*                                                                     00180000
//*    NOTE:  DDF MUST BE UP FOR STEP PH03CS06 TO EXECUTE               00190000
//*                                                                     00200000
//*    RUN THIS JOB ANYTIME AFTER PHASE 2.                              00210000
//*                                                                     00220000
//*                                                                     00230000
//*                                                                     00240000
//*  CHANGE ACTIVITY =                                                  00250000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00251000
//*               Add SET CURRENT SQLID                                 00251101
//*                                                                     00252000
//*                                                                     00260000
//********************************************************************* 00270000
//*                                                                     00280000
//JOBLIB   DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                             00290000
//*                                                                     00300000
//*                                                                     00310000
//* STEP  1: PREPARE THE COBOL CAF INTERFACE                            00320000
//*                                                                     00330000
//PH03CS01 EXEC DSNHICOB,MEM=DSN8CC,                                    00340000
//         COND=(4,LT),                                                 00350000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00360000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00370000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')        00380000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CC),                      00390000
//             DISP=SHR                                                 00400000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00410000
//             DISP=SHR                                                 00420000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8CC),                          00430000
//             DISP=SHR                                                 00440000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CC),                       00450000
//             DISP=SHR                                                 00460000
//LKED.SYSIN   DD *                                                     00470000
  INCLUDE SYSLIB(DSNALI)                                                00480000
//*                                                                     00490000
//* STEP  2: PREPARE THE CONNECTION MANAGER                             00500000
//*                                                                     00510000
//PH03CS02 EXEC DSNHICOB,MEM=DSN8SCM,                                   00520000
//         COND=(4,LT),                                                 00530000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00540000
```

```
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00550000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')        00560000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SCM),                     00570000
//             DISP=SHR                                                 00580000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00590000
//             DISP=SHR                                                 00600000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8SCM),                         00610000
//             DISP=SHR                                                 00620000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SCM),                      00630000
//             DISP=SHR                                                 00640000
//*                                                                     00650000
//* STEP  3: PREPARE THE TELEPHONE APPLICATION                          00660000
//*                                                                     00670000
//PH03CS03 EXEC DSNHICOB,MEM=DSN8SC3,                                   00680000
//         COND=(4,LT),                                                 00690000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00700000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00710000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')        00720000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SC3),                     00730000
//             DISP=SHR                                                 00740000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00750000
//             DISP=SHR                                                 00760000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8SC3),                         00770000
//             DISP=SHR                                                 00780000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SC3),                      00790000
//             DISP=SHR                                                 00800000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               00810000
//             DISP=SHR                                                 00820000
//LKED.SYSIN   DD *                                                     00830000
     INCLUDE SYSLIB(DSNALI)                                             00840000
     INCLUDE RUNLIB(DSN8MCG)                                            00850000
//*                                                                     00860000
//* STEP  4: PREPARE THE REMOTE ORGANIZATION APPLICATION                00870000
//*                                                                     00880000
//PH03CS04 EXEC DSNHICOB,MEM=DSN8HC3,                                   00890000
//         COND=(4,LT),                                                 00900000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00910000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00920000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')        00930000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8HC3),                     00940000
//             DISP=SHR                                                 00950000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00960000
//             DISP=SHR                                                 00970000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8HC3),                         00980000
//             DISP=SHR                                                 00990000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8HC3),                      01000000
//             DISP=SHR                                                 01010000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               01020000
//             DISP=SHR                                                 01030000
//LKED.SYSIN   DD *                                                     01040000
     INCLUDE SYSLIB(DSNALI)                                             01050000
     INCLUDE RUNLIB(DSN8MCG)                                            01060000
//*                                                                     01070000
//* STEP  5: BIND THE TELEPHONE APPLICATION PROGRAM                     01080000
//*                                                                     01090000
//PH03CS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01100000
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                          01110000
//SYSTSPRT DD SYSOUT=*                                                  01120000
//SYSPRINT DD SYSOUT=*                                                  01130000
//SYSUDUMP DD SYSOUT=*                                                  01140000
//SYSOUT   DD SYSOUT=*                                                  01150000
//SYSIN    DD *                                                         01160000
 SET CURRENT SQLID = 'SYSADM';                                          01161001
 GRANT BIND, EXECUTE ON PLAN DSN8SC!!                                   01170002
   TO PUBLIC;                                                           01171002
//SYSTSIN DD *                                                          01180000
 DSN SYSTEM(DSN)                                                        01190000
 BIND PACKAGE(DSN8SC!!) MEMBER(DSN8SC3) +                               01200000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01210000
 BIND PLAN(DSN8SC!!) PKLIST(DSN8SC!!.*) +                               01211000
      ACTION(REPLACE) RETAIN +                                          01211100
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01212000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                01220000
      LIB('DSN!!0.RUNLIB.LOAD')                                         01230000
//*                                                                     01240000
//* STEP  6: BIND THE REMOTE ORGANIZATION APPLICATION                   01250000
//*                                                                     01260000
//PH03CS06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01270000
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                          01280000
//SYSTSPRT DD SYSOUT=*                                                  01290000
//SYSPRINT DD SYSOUT=*                                                  01300000
//SYSUDUMP DD SYSOUT=*                                                  01310000
```

```
//SYSOUT   DD SYSOUT=*                                                  01320000
//SYSIN    DD *                                                         01330000
 SET CURRENT SQLID = 'SYSADM';                                          01331001
 GRANT BIND, EXECUTE ON PLAN DSN8HC!!                                   01340002
   TO PUBLIC;                                                           01341002
//SYSTSIN DD *                                                          01350000
 DSN SYSTEM(DSN)                                                        01360000
 BIND PACKAGE(DSN8HC!!) MEMBER (DSN8HC3) -                              01370000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01380000
 BIND PACKAGE(SAMPLOC.DSN8HC!!) MEMBER(DSN8HC3) -                       01390000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01400000
 BIND PLAN(DSN8HC!!) -                                                  01410000
      PKLIST(DSN8HC!!.*,SAMPLOC.DSN8HC!!.*) -                           01411000
      ACTION(REPLACE) RETAIN +                                          01412000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01420000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                01430000
      LIB('DSN!!0.RUNLIB.LOAD')                                         01440000
 END                                                                    01450000
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6

RUN THIS JOB AT THE REMOTE LOCATION TO UPDATE THE SAMPLE LOCATION IN DEPARTMENT TABLE RUN THIS JOB ANYTIME AFTER PHASE 3.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ6                                                     00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 REMOTE UNIT OF WORK SAMPLE APPLICATION      00040000
//*                     PHASE 6                                         00050000
//*                                                                     00060000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00070000
//*    5615-DB2                                                         00080001
//*    (C) COPYRIGHT 1982, 2013 IBM CORP.  ALL RIGHTS RESERVED.         00090001
//*                                                                     00100000
//*    STATUS = VERSION 11                                              00110001
//*                                                                     00120000
//*  FUNCTION = RUN THIS JOB AT THE REMOTE LOCATION TO UPDATE THE       00130000
//*             SAMPLE LOCATION IN DEPARTMENT TABLE                     00140000
//*                                                                     00150000
//*        RUN THIS JOB ANYTIME AFTER PHASE 3.                          00160000
//*                                                                     00170000
//*  CHANGE ACTIVITY =                                                  00180000
//*    11/07/2012 ADD SET CURRENT SQLID           DN1651_INST1 / DN1651 00180100
//*    05/17/2013 FIX COPYRIGHT STATEMENT                  49779_077_724 00180201
//*                                                                     00181000
//********************************************************************* 00190000
//*                                                                     00200000
//JOBLIB   DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                             00210000
//*                                                                     00220000
//* STEP  1: UPDATE SAMPLE LOCATIONS IN DEPARTMENT TABLE                00230000
//*                                                                     00240000
//PH06S01  EXEC PGM=IKJEFT01,DYNAMNBR=20                                00250000
//SYSTSPRT DD SYSOUT=*                                                  00260000
//SYSPRINT DD SYSOUT=*                                                  00270000
//SYSUDUMP DD SYSOUT=*                                                  00280000
//SYSOUT   DD SYSOUT=*                                                  00290000
//SYSTSIN DD *                                                          00300000
 DSN SYSTEM(DSN)                                                        00310000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                00320000
      LIB('DSN!!0.RUNLIB.LOAD')                                         00330000
//SYSIN    DD *                                                         00340000
 SET CURRENT SQLID = 'SYSADM';                                          00341000
 UPDATE DEPT SET LOCATION = 'SAMPLOC' WHERE DEPTNO = 'F22';             00350000
 UPDATE DEPT SET LOCATION = 'THISLOCN' WHERE LOCATION = ' ';            00360000
//*                                                                     00370000
```

Related reference

“Sample applications in TSO” on page 1013

A set of Db2 sample applications run in the TSO environment.

DSNTEJ3P

THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH PL/I.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ3P                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 3                                         00050000
//*                     PL/I, ISPF, CAF                                 00060000
//*                                                                     00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00080000
//*    5605-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00100000
//*                                                                     00110000
//*    STATUS = VERSION 11                                              00120001
//*                                                                     00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE      00140000
//*             APPLICATIONS AT SITES WITH PL/I.  IT PREPARES THE       00150000
//*             PL/I ISPF CAF TELEPHONE PROGRAM.                        00160000
//*                                                                     00170000
//*        RUN THIS JOB ANYTIME AFTER PHASE 2.                          00180000
//*                                                                     00190000
//*  CHANGE ACTIVITY =                                                  00200000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00201000
//*               ADD SET CURRENT SQLID                                 00202002
//*                                                                     00210000
//********************************************************************* 00220000
//*                                                                     00230000
//JOBLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                              00240000
//        DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN                           00250000
//*                                                                     00260000
//* STEP  1: PREPARE THE ISPF CAF CONNECTION MANAGER                    00270000
//*                                                                     00280000
//PH03PS01 EXEC DSNHPLI,MEM=DSN8SPM,                                    00290000
//           COND=(4,LT),                                               00300000
//           PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',      00310000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            00320000
//           'LIMITS(EXTNAME(7)),OPTIONS')                              00330000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8SPM),                         00340000
//             DISP=SHR                                                 00350000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SPM),                     00360000
//             DISP=SHR                                                 00370000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00380000
//             DISP=SHR                                                 00390000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SPM),                      00400000
//             DISP=SHR                                                 00410000
//LKED.SYSIN   DD *                                                     00420000
     ENTRY CEESTART                                                     00430000
//*                                                                     00440000
//* STEP  2: PREPARE THE ISPF CAF TELEPHONE APPLICATION                 00450000
//*                                                                     00460000
//PH03PS02 EXEC DSNHPLI,MEM=DSN8SP3,                                    00470000
//           COND=(4,LT),                                               00480000
//           PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',      00490000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            00500000
//           'LIMITS(EXTNAME(7)),OPTIONS')                              00510000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8SP3),                         00520000
//             DISP=SHR                                                 00530000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SP3),                     00540000
//             DISP=SHR                                                 00550000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00560000
//             DISP=SHR                                                 00570000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SP3),                      00580000
//             DISP=SHR                                                 00590000
//LKED.RUNLIB  DD DSN=DSN!!0.RUNLIB.LOAD,                               00600000
//             DISP=SHR                                                 00610000
//LKED.SYSIN   DD *                                                     00620000
     INCLUDE SYSLIB(DSNALI)                                             00630000
     INCLUDE RUNLIB(DSN8MPG)                                            00640000
//*                                                                     00650000
//* STEP  3: BIND THE ISPF CAF TELEPHONE APPLICATION                    00660000
//*                                                                     00670000
//PH03PS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00680000
//DBRMLIB  DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                          00690000
//SYSTSPRT DD SYSOUT=*                                                  00700000
//SYSPRINT DD SYSOUT=*                                                  00710000
```

```
//SYSUDUMP DD SYSOUT=*                                                  00720000
//SYSIN   DD *                                                          00730000
   SET CURRENT SQLID = 'SYSADM';                                        00731002
   GRANT BIND, EXECUTE ON PLAN DSN8SP!!                                 00740003
     TO PUBLIC;                                                         00741003
//SYSTSIN DD *                                                          00750000
 DSN SYSTEM(DSN)                                                        00760000
 BIND PACKAGE(DSN8SP!!) MEMBER(DSN8SP3) +                               00770000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00771000
 BIND PLAN(DSN8SP!!) PKLIST(DSN8SP!!.*) +                               00772000
      ACTION(REPLACE) RETAIN +                                          00773000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         00780000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                 00790000
      LIB('DSN!!0.RUNLIB.LOAD')                                         00800000
 END                                                                    00810000
//*                                                                     00820000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2A

THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE APPLICATIONS.

```
//********************************************************************* 00000100
//*  NAME = DSNTEJ2A                                                    00000200
//*                                                                     00000300
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00000400
//*                     PHASE 2                                         00000500
//*                     ASSEMBLER                                       00000600
//*                                                                     00000700
//*    Licensed Materials - Property of IBM                             00000800
//*    5605-DB2                                                         00000900
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.         00001000
//*                                                                     00001100
//*    STATUS = Version 11                                              00001200
//*                                                                     00001300
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE      00001400
//*             APPLICATIONS.  IT PREPARES AND RUNS THE SAMPLE          00001500
//*             ASSEMBLER BATCH TABLE UNLOAD PROGRAM                    00001600
//*                                                                     00001700
//*             THIS JOB IS RUN AFTER PHASE 1.                          00001800
//*                                                                     00001900
//*  NOTICE =                                                           00002000
//*    THIS SAMPLE JOB USES DB2 UTILITIES. SOME UTILITY FUNCTIONS ARE   00002100
//*    ELEMENTS OF SEPARATELY ORDERABLE PRODUCTS.  SUCCESSFUL USE OF    00002200
//*    A PARTICULAR SAMPLE JOB MAY BE DEPENDENT UPON THE OPTIONAL       00002300
//*    PRODUCT BEING LICENSED AND INSTALLED IN YOUR ENVIRONMENT.        00002400
//*                                                                     00002500
//*  CHANGE ACTIVITY =                                                  00002600
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00002700
//*               Add SET CURRENT SQLID                                 00002800
//*                                                                     00002900
//********************************************************************* 00003000
//*                                                                     00003100
//JOBLIB  DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00003200
//*                                                                     00003300
//* PRECOMPILE, ASSEMBLE, AND LINK EDIT THE UNLOAD PROGRAM              00003400
//*                                                                     00003500
//PREPUNL EXEC DSNHASM,MEM=DSNTIAUL,                                    00003600
//         PARM.PC='HOST(ASM),STDSQL(NO),VERSION(AUTO)',                00003700
//         PARM.ASM='RENT,OBJECT,NODECK',                               00003800
//         PARM.LKED='RENT,XREF,AMODE=ANY,RMODE=24'                     00003900
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSNTIAUL),                    00004000
//             DISP=SHR                                                 00004100
//PC.SYSLIB    DD DSN=DSN!!0.SDSNSAMP,                                  00004200
//             DISP=SHR                                                 00004300
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSNTIAUL),                        00004400
//             DISP=SHR                                                 00004500
//ASM.SYSLIB   DD                                                       00004600
//             DD DSN=DSN!!0.SDSNMACS,                                  00004700
//             DISP=SHR                                                 00004800
//             DD DSN=DSN!!0.SDSNSAMP,                                  00004900
//             DISP=SHR                                                 00005000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTIAUL),                     00005100
//             DISP=SHR                                                 00005200
//LKED.SYSIN   DD *                                                     00005300
```

```
  INCLUDE SYSLIB(DSNELI)
  NAME DSNTIAUL(R)
//*
//* BIND THE UNLOAD PROGRAM AND GRANT EXECUTE AUTHORITY
//*
//BINDUNL EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,
//             DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DSN)
  BIND PACKAGE(DSNTIB!!) MEM(DSNTIAUL) +
       CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC) +
       LIB('DSN!!0.DBRMLIB.DATA')
  BIND PLAN(DSNTIB!!) PKLIST(DSNTIB!!.*) +
       ACTION(REPLACE) RETAIN +
       CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC)
  RUN PROGRAM(DSNTIAD)  PLAN(DSNTIA!!) +
       LIB('DSN!!0.RUNLIB.LOAD')
  END
//SYSIN    DD  *
   SET CURRENT SQLID = 'SYSADM';
   GRANT EXECUTE ON PLAN DSNTIB!!
     TO PUBLIC;
//*
//*        DELETE DATA SETS, DROP TABLES TO ALLOW RERUNS
//*
//DELETE  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  DELETE 'DSN!!0.DSN8UNLD.SYSREC00'
  DELETE 'DSN!!0.DSN8UNLD.SYSREC01'
  DELETE 'DSN!!0.DSN8UNLD.SYSPUNCH'
  DSN SYSTEM(DSN)
  RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) PARM('RC0') -
      LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
  SET CURRENT SQLID = 'SYSADM';
  DROP TABLE DSN8!!0.NEWDEPT ;
  DROP TABLE DSN8!!0.NEWPHONE ;
  DROP DATABASE DSN8D!!U  ;
  DROP STOGROUP DSN8G!!U  ;
  COMMIT;
//*
//*        CREATE NEW TABLES
//*
//CREATE  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
 SET CURRENT SQLID = 'SYSADM';
 CREATE STOGROUP DSN8G!!U
        VOLUMES (DSNV01)
        VCAT DSNC!!0;

 CREATE DATABASE DSN8D!!U
        STOGROUP DSN8G!!U
        CCSID EBCDIC;

 CREATE TABLE DSN8!!0.NEWDEPT
              (DEPTNO         CHAR(3)        NOT NULL,
               DEPTNAME       VARCHAR(36)    NOT NULL,
               MGRNO          CHAR(6)                ,
               ADMRDEPT       CHAR(3)        NOT NULL,
               LOCATION       CHAR(16))
         IN DATABASE DSN8D!!U
         CCSID EBCDIC;

 CREATE TABLE  DSN8!!0.NEWPHONE
              (LASTNAME       VARCHAR(15)    NOT NULL,
               FIRSTNAME      VARCHAR(12)    NOT NULL,
               MIDDLEINITIAL  CHAR(1)        NOT NULL,
               PHONENUMBER    CHAR(4)                ,
```

```
                 EMPLOYEENUMBER CHAR(6)        NOT NULL,                00013600
                 DEPTNUMBER     CHAR(3)        NOT NULL,                00013700
                 DEPTNAME       VARCHAR(36)    NOT NULL   )             00013800
          IN DATABASE DSN8D!!U                                          00013900
          CCSID EBCDIC;                                                 00014000
//*                                                                     00014100
//*        RUN UNLOAD PROGRAM                                           00014200
//*                                                                     00014300
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                     00014400
//SYSTSPRT  DD  SYSOUT=*                                                00014500
//SYSTSIN   DD  *                                                       00014600
 DSN SYSTEM(DSN)                                                        00014700
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB!!) PARMS('SQL') -                   00014800
      LIB('DSN!!0.RUNLIB.LOAD')                                         00014900
//SYSPRINT  DD SYSOUT=*                                                 00015000
//SYSUDUMP  DD SYSOUT=*                                                 00015100
//SYSREC00  DD DSN=DSN!!0.DSN8UNLD.SYSREC00,                            00015200
//             DISP=(,CATLG),                                           00015300
//             UNIT=SYSDA,                                              00015400
//             SPACE=(1024,(10,10))                                     00015500
//SYSREC01  DD DSN=DSN!!0.DSN8UNLD.SYSREC01,                            00015600
//             DISP=(,CATLG),                                           00015700
//             UNIT=SYSDA,                                              00015800
//             SPACE=(1024,(10,10))                                     00015900
//SYSPUNCH  DD DSN=DSN!!0.DSN8UNLD.SYSPUNCH,                            00016000
//             DISP=(,CATLG),                                           00016100
//             UNIT=SYSDA,                                              00016200
//             SPACE=(800,(15,15))                                      00016300
//SYSIN     DD *                                                        00016400
    SET CURRENT SQLID = 'SYSADM';                                       00016500
    LOCK TABLE DSN8!!0.DEPT IN SHARE MODE;                              00016600
    SELECT * FROM DSN8!!0.DEPT;                                         00016700
    SELECT * FROM DSN8!!0.VPHONE;                                       00016800
//*                                                                     00016900
//*       EDIT THE OUTPUT FROM THE PROGRAM                              00017000
//*                                                                     00017100
//EDIT    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=((4,LT),(4,LE,UNLOAD))     00017200
//SYSTSPRT DD  SYSOUT=*                                                 00017300
//SYSTSIN  DD  *                                                        00017400
 EDIT 'DSN!!0.DSN8UNLD.SYSPUNCH' DATA NONUM                             00017500
 CHANGE * 30 /DSN8!!0.DEPT/DSN8!!0.NEWDEPT/                             00017600
 CHANGE * 30 /DSN8!!0.VPHONE/DSN8!!0.NEWPHONE/                          00017700
 TOP                                                                    00017800
 LIST * 999                                                             00017900
 END SAVE                                                               00018000
//*                                                                     00018100
//*        RUN LOAD UTILITY TO LOAD TABLES                              00018200
//*                                                                     00018300
//LOAD    EXEC DSNUPROC,PARM='DSN,DSNTEX',                              00018400
//        COND=((4,LT),(4,LE,UNLOAD))                                   00018500
//DSNTRACE DD  SYSOUT=*                                                 00018600
//SORTLIB  DD  DSN=SYS1.SORTLIB,DISP=SHR                                00018700
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)                  00018800
//SORTWK02 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)                  00018900
//SORTWK03 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)                  00019000
//SORTWK04 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)                  00019100
//SYSREC00 DD  DSN=DSN!!0.DSN8UNLD.SYSREC00,                            00019200
//             DISP=(OLD,KEEP)                                          00019300
//SYSREC01 DD  DSN=DSN!!0.DSN8UNLD.SYSREC01,                            00019400
//             DISP=(OLD,KEEP)                                          00019500
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)                  00019600
//SYSIN    DD  DSN=DSN!!0.DSN8UNLD.SYSPUNCH,                            00019700
//             DISP=(OLD,KEEP)                                          00019800
//*                                                                     00019900
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ1P

THIS JCL PERFORMS THE PHASE 1 SETUP FOR SAMPLE APPLICATIONS AT SITES WITH PL/I.

```
//********************************************************************* 00010002
//*  NAME = DSNTEJ1P                                                    00020002
//*                                                                     00030002
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040002
//*                     PHASE 1                                         00050002
```

```
//*                     PL/I
//*
//*    Licensed Materials - Property of IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.
//*
//*    STATUS = Version 11
//*
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 1 SETUP FOR SAMPLE
//*             APPLICATIONS AT SITES WITH PL/I.
//*
//*         THIS JOB IS RUN AFTER DSNTEJ1.
//*
//*  CHANGE ACTIVITY =
//*    03/08/02 - ADD COMPILER OPTION TO SUPPORT PL/I POINTERADD
//*               FUNCTION.  ADD STACK(,,ANY) RUN TIME OPTION TO
//*               ALLOW PROGRAM VARIABLES TO BE STORED ABOVE THE
//*               16M LINE.
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651
//*               Add SET CURRENT SQLID
//*
//*********************************************************************
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//         DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//*        STEP 1 : PREPARE DSNTEP2 FOR EXECUTION
//PH01PS01 EXEC DSNHPLI,MEM=DSNTEP2,
//         PARM.PC=('HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)',
//             TWOPASS,'VERSION(AUTO)'),
//         PARM.PLI=(NOPT,'MAR(2,72,0)',GS,OBJ,S,
//         'LIMITS(FIXEDBIN(31,63))','LANGLVL(SPROG)',OFFSET)
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSNTEP2),
//             DISP=SHR
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSNTEP2),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTEP2),
//             DISP=SHR
//LKED.SYSIN   DD *
     INCLUDE SYSLIB(DSNELI)
//*
//*        STEP 2 : BIND AND RUN PROGRAM DSNTEP2, TO
//*                 PRINT THE TABLES
//PH01PS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,
//             DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 BIND PACKAGE (DSNTEP2) MEMBER(DSNTEP2) +
      CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
 BIND PLAN(DSNTEP!!)  PKLIST(DSNTEP2.*) +
      ACTION(REPLACE) RETAIN +
      CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +
      LIB('DSN!!0.RUNLIB.LOAD') +
      PARMS('/ALIGN(MID)')
 END
//*
//SYSIN    DD *
    SET CURRENT SQLID = 'SYSADM';
    GRANT EXECUTE, BIND ON PLAN DSNTEP!!
      TO PUBLIC;
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
      WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
      SEX, BIRTHDATE, SALARY, BONUS, COMM,
      SALARY+BONUS+COMM AS TOTAL_SALARY
     FROM EMP
     ORDER BY TOTAL_SALARY;
    SELECT * FROM DEPT;
    SELECT * FROM ACT;
    SELECT * FROM EMPPROJACT;
    SELECT * FROM PROJ;
    SELECT * FROM PROJACT;
//*
//*
//*        STEP 3 : PREPARE DSNTEP4 FOR EXECUTION
//PH01PS03 EXEC DSNHPLI,MEM=DSNTEP4,COND=(4,LT),
//         PARM.PC=('HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)',
```

```
//              TWOPASS,'VERSION(AUTO)'),                               00830000
//         PARM.PLI=(NOPT,'MAR(2,72,0)',GS,OBJ,S,                       00840000
//         'LIMITS(FIXEDBIN(31,63))','LANGLVL(SPROG)',OFFSET)           00850000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSNTEP4),                         00860000
//              DISP=SHR                                                00870000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSNTEP4),                     00880000
//              DISP=SHR                                                00890000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00900000
//              DISP=SHR                                                00910000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTEP4),                      00920000
//              DISP=SHR                                                00930000
//LKED.SYSIN   DD *                                                     00940000
     INCLUDE SYSLIB(DSNELI)                                             00950000
//*                                                                     00960000
//*        STEP 4 : BIND AND RUN PROGRAM DSNTEP4, TO                    00970000
//*                 PRINT THE TABLES                                    00980000
//PH01PS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00990000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,                                 01000000
//              DISP=SHR                                                01010000
//SYSTSPRT DD  SYSOUT=*                                                 01020000
//SYSPRINT DD  SYSOUT=*                                                 01030000
//SYSUDUMP DD  SYSOUT=*                                                 01040000
//SYSTSIN  DD  *                                                        01050000
 DSN SYSTEM(DSN)                                                        01060000
 BIND PACKAGE (DSNTEP4) MEMBER(DSNTEP4) +                               01070000
      CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)                 01080000
 BIND PLAN(DSNTP4!!)  PKLIST(DSNTEP4.*) +                               01090000
      ACTION(REPLACE) RETAIN +                                          01091000
      CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)            01100000
 RUN  PROGRAM(DSNTEP4) PLAN(DSNTP4!!) +                                 01110000
      LIB('DSN!!0.RUNLIB.LOAD') +                                       01120000
      PARMS('/ALIGN(MID)')                                              01130000
 END                                                                    01140000
//*                                                                     01150000
//SYSIN    DD *                                                         01160000
    SET CURRENT SQLID = 'SYSADM';                                       01161003
    GRANT EXECUTE, BIND ON PLAN DSNTP4!!                                01170004
      TO PUBLIC;                                                        01171004
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,                          01180000
      WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,                        01190000
      SEX, BIRTHDATE, SALARY, BONUS, COMM,                              01200000
      SALARY+BONUS+COMM AS TOTAL_SALARY                                 01210000
     FROM EMP                                                           01220000
     ORDER BY TOTAL_SALARY;                                             01230000
    SELECT * FROM DEPT;                                                 01240000
    SELECT * FROM ACT;                                                  01250000
    SELECT * FROM EMPPROJACT;                                           01260000
    SELECT * FROM PROJ;                                                 01270000
    SELECT * FROM PROJACT;                                              01280000
//*                                                                     01290000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ1L

THIS JCL CREATES THE DSNTEP2 LOAD MODULE FROM THE SHIPPED OBJECT DECK, DSNTEP2L, AND LINKS THE PACKAGE AND PLAN FOR THIS VERSION OF DSNTEP2.

```
//********************************************************************** 00000100
//*  NAME = DSNTEJ1L                                                    00000200
//*                                                                     00000300
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00000400
//*                     PHASE 1                                         00000500
//*                     L/E                                             00000600
//*                                                                     00000700
//*    Licensed Materials - Property of IBM                             00000800
//*    5605-DB2                                                         00000900
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.         00001000
//*                                                                     00001100
//*    STATUS = Version 11                                              00001200
//*                                                                     00001300
//*  FUNCTION = THIS JCL CREATES THE DSNTEP2 LOAD MODULE FROM THE       00001400
//*             SHIPPED OBJECT DECK, DSNTEP2L, AND LINKS THE            00001500
//*             PACKAGE AND PLAN FOR THIS VERSION OF DSNTEP2.           00001600
//*                                                                     00001700
//*           = THIS JCL ALSO CREATES THE DSNTEP4 LOAD MODULE FROM      00001800
```

```
//*             THE SHIPPED OBJECT DECK, DSNTEP4L, AND LINKS THE        00001900
//*             PACKAGE AND PLAN FOR THIS VERSION OF DSNTEP4.           00002000
//*                                                                     00002100
//*         THIS JOB IS RUN AFTER DSNTEJ1.                              00002200
//*                                                                     00002300
//*   NOTE: IF YOU RUN THIS JOB, YOU DO NOT NEED TO RUN THE SAMPLE      00002400
//*         JOB DSNTEJ1P EXCEPT TO PREPARE CUSTOMIZED VERSIONS OF       00002500
//*         THE DSNTEP2 AND DSNTEP4 SOURCE CODE (YOU NEED A PL/I        00002600
//*         COMPILER TO RUN DSNTEJ1P SUCCESSFULLY).                     00002700
//*                                                                     00002800
//*  CHANGE ACTIVITY =                                                  00002900
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00003000
//*               Add SET CURRENT SQLID                                 00003100
//*                                                                     00003200
//******************************************************************** 00003300
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00003400
//         DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN                          00003500
//*                                                                     00003600
//*        STEP 1 : CREATE DSNTEP2 LOADMOD FROM DSNTEP2L OBJECT DECK    00003700
//*                                                                     00003800
//PH01LS01 EXEC PGM=IEWL,PARM='XREF'                                    00003900
//SYSLIB   DD  DISP=SHR,DSN=CEE.V!R!M!.SCEELKED                         00004000
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00004100
//SDSNSAMP DD  DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSNTEP2L)                   00004200
//SYSLMOD  DD  DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD(DSNTEP2)                 00004300
//SYSPRINT DD  SYSOUT=*                                                 00004400
//SYSUDUMP DD  SYSOUT=*                                                 00004500
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))                          00004600
//SYSLIN   DD  *                                                        00004700
     INCLUDE SDSNSAMP(DSNTEP2L)                                         00004800
     INCLUDE SYSLIB(DSNELI)                                             00004900
     NAME DSNTEP2(R)                                                    00005000
//*                                                                     00005100
//*        STEP 2 : BIND AND RUN PROGRAM DSNTEP2, TO                    00005200
//*                 PRINT THE TABLES                                    00005300
//*                                                                     00005400
//PH01LS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00005500
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNSAMP                             00005600
//SYSTSPRT DD  SYSOUT=*                                                 00005700
//SYSPRINT DD  SYSOUT=*                                                 00005800
//SYSUDUMP DD  SYSOUT=*                                                 00005900
//SYSTSIN  DD  *                                                        00006000
 DSN SYSTEM(DSN)                                                        00006100
 BIND PACKAGE (DSNTEP2) MEMBER(DSN@EP2L) +                              00006200
      CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)                 00006300
 BIND PLAN(DSNTEP!!)  PKLIST(DSNTEP2.*) +                               00006400
      ACTION(REPLACE) RETAIN +                                          00006500
      CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)            00006600
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +                                 00006700
      LIB('DSN!!0.RUNLIB.LOAD') +                                       00006800
      PARMS('/ALIGN(MID)')                                              00006900
 END                                                                    00007000
//*                                                                     00007100
//SYSIN    DD *                                                         00007200
    SET CURRENT SQLID = 'SYSADM';                                       00007300
    GRANT EXECUTE, BIND ON PLAN DSNTEP!!                                00007400
      TO PUBLIC;                                                        00007500
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,                          00007600
      WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,                        00007700
      SEX, BIRTHDATE, SALARY, BONUS, COMM,                              00007800
      SALARY+BONUS+COMM AS TOTAL_SALARY                                 00007900
     FROM EMP                                                           00008000
     ORDER BY TOTAL_SALARY;                                             00008100
    SELECT * FROM DEPT;                                                 00008200
    SELECT * FROM ACT;                                                  00008300
    SELECT * FROM EMPPROJACT;                                           00008400
    SELECT * FROM PROJ;                                                 00008500
    SELECT * FROM PROJACT;                                              00008600
//*                                                                     00008700
//*        STEP 3 : CREATE DSNTEP4 LOADMOD FROM DSNTEP4L OBJECT DECK    00008800
//*                                                                     00008900
//PH01LS03 EXEC PGM=IEWL,COND=(4,LT),PARM='XREF'                        00009000
//SYSLIB   DD  DISP=SHR,DSN=CEE.V!R!M!.SCEELKED                         00009100
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00009200
//SDSNSAMP DD  DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSNTEP4L)                   00009300
//SYSLMOD  DD  DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD(DSNTEP4)                 00009400
//SYSPRINT DD  SYSOUT=*                                                 00009500
//SYSUDUMP DD  SYSOUT=*                                                 00009600
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))                          00009700
//SYSLIN   DD  *                                                        00009800
     INCLUDE SDSNSAMP(DSNTEP4L)                                         00009900
     INCLUDE SYSLIB(DSNELI)                                             00010000
```

```
     NAME DSNTEP4(R)
//*
//*        STEP 4 : BIND AND RUN PROGRAM DSNTEP4, TO
//*                 PRINT THE TABLES
//*
//PH01LS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNSAMP
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 BIND PACKAGE (DSNTEP4) MEMBER(DSN@EP4L) +
      CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
 BIND PLAN(DSNTP4!!)  PKLIST(DSNTEP4.*) +
      ACTION(REPLACE) RETAIN +
      CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
 RUN  PROGRAM(DSNTEP4) PLAN(DSNTP4!!) +
      LIB('DSN!!0.RUNLIB.LOAD') +
      PARMS('/ALIGN(MID)')
 END
//*
//SYSIN    DD *
    SET CURRENT SQLID = 'SYSADM';
    GRANT EXECUTE, BIND ON PLAN DSNTP4!!
      TO PUBLIC;
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
      WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
      SEX, BIRTHDATE, SALARY, BONUS, COMM,
      SALARY+BONUS+COMM AS TOTAL_SALARY
     FROM EMP
     ORDER BY TOTAL_SALARY;
    SELECT * FROM DEPT;
    SELECT * FROM ACT;
    SELECT * FROM EMPPROJACT;
    SELECT * FROM PROJ;
    SELECT * FROM PROJACT;
//*
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6P

THIS JCL EXECUTES THE PHASE 6 STORED PROCEDURE SAMPLE APPLICATION.

```
//********************************************************************
//*  NAME = DSNTEJ6P
//*
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//*
//*                     PHASE 6
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5635-DB2
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 11
//*
//*  FUNCTION = THIS JCL EXECUTES THE PHASE 6 STORED PROCEDURE
//*             SAMPLE APPLICATION.
//*
//*  DEPENDENCIES:
//*  (1) RUN SAMPLE JOB DSNTEJ6S AT THE SERVER SITE BEFORE RUNNING THIS
//*      JOB; DSNTEJ6S PREPARES THE SAMPLE STORED PROC W/O RESULT SET
//*  (2) RUN THIS JOB AT THE CLIENT SITE
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651
//*               ADD SET CURRENT SQLID
//*
//********************************************************************
//JOBLIB   DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
//         DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//         DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//         DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//*
```

```
//********************************************************************  00290000
//*  STEP 1:  PRE-COMPILE, COMPILE, AND LINK-EDIT THE CALLING PROGRAM   00300000
//********************************************************************  00310000
//PH06PS01  EXEC DSNHPLI,MEM=DSN8EP1,                                   00320000
//          PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)'         00330000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8EP1),                         00340000
//              DISP=SHR                                                00350000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EP1),                     00360000
//              DISP=SHR                                                00370000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00380000
//              DISP=SHR                                                00390000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EP1),                      00400000
//              DISP=SHR                                                00410000
//LKED.SYSIN DD  *                                                      00420000
 INCLUDE SYSLIB(DSNELI)                                                 00430000
 INCLUDE SYSLIB(DSNTIAR)                                                00440000
//********************************************************************  00450000
//*  STEP 2:  BIND THE CALLING PROGRAM PACKAGE                          00460000
//********************************************************************  00470000
//PH06PS02   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                  00480000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                 00490000
//              DISP=SHR                                                00500000
//SYSTSPRT DD SYSOUT=*                                                  00510000
//SYSPRINT DD SYSOUT=*                                                  00520000
//SYSUDUMP DD SYSOUT=*                                                  00530000
//SYSIN    DD *                                                         00540000
 SET CURRENT SQLID = 'SYSADM';                                          00541001
 GRANT BIND, EXECUTE ON PLAN DSN8EP1                                    00550002
   TO PUBLIC;                                                           00551002
//SYSTSIN  DD  *                                                        00560000
 DSN SYSTEM(DSN)                                                        00570000
 BIND PACKAGE(DSN8EP!!) MEMBER(DSN8EP1) -                               00580000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00590000
 BIND PACKAGE(SAMPLOC.DSN8EP!!) -                                       00600000
   MEMBER(DSN8EP1) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)   00610000
 BIND PLAN(DSN8EP1) -                                                   00620000
 PKLIST(DSN8EP!!.DSN8EP1, SAMPLOC.DSN8EP!!.DSN8EP1) -                   00630000
   ACTION(REPLACE) RETAIN +                                             00631000
   ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                            00640000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                                00650000
     LIB('DSN!!0.RUNLIB.LOAD')                                          00660000
//********************************************************************  00670000
//*  STEP 3:  EXECUTE THE STORED PROCEDURE                              00680000
//********************************************************************  00690000
//PH06PS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00700000
//SYSTSPRT DD  SYSOUT=*                                                 00710000
//SYSPRINT DD  SYSOUT=*                                                 00720000
//SYSUDUMP DD  SYSOUT=*                                                 00730000
//SYSTSIN  DD  *                                                        00740000
 DSN SYSTEM(DSN)                                                        00750000
 RUN PROGRAM(DSN8EP1) PLAN(DSN8EP1) PARMS('/SAMPLOC')                   00760000
 END                                                                    00770000
//SYSIN    DD *                                                         00780000
  -DISPLAY ARCHIVE;                                                     00790000
  -DISPLAY THREAD(*) DETAIL;                                            00800000
//*                                                                     00810000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6S

THIS JCL PREPARES THE SAMPLE STORED PROCEDURE.

```
//********************************************************************  00010000
//*  NAME = DSNTEJ6S                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                                                                     00050000
//*                     PHASE 6                                         00060000
//*                                                                     00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00080000
//*    5635-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.         00100000
//*                                                                     00110000
//*    STATUS = VERSION 9                                               00120000
//*                                                                     00130000
```

```
//*  FUNCTION = THIS JCL PREPARES THE SAMPLE STORED PROCEDURE.          00140000
//*                                                                     00150000
//*  DEPENDENCIES:                                                      00160000
//*  (1) RUN THIS JOB AT THE SERVER SITE BEFORE RUNNING SAMPLE JOB      00170000
//*      DSNTEJ6P AT THE CLIENT SITE                                    00180000
//*                                                                     00190000
//*  CHANGE ACTIVITY =                                                  00200000
//*    11/07/2012 ADD SET CURRENT SQLID           DN1651_INST1 / DN1651 00200101
//*                                                                     00201000
//********************************************************************  00210000
//JOBLIB   DD DISP=SHR,DSN=DSN!!0.SDSNEXIT                              00220000
//         DD DISP=SHR,DSN=DSN!!0.SDSNLOAD                              00230000
//*                                                                     00240000
//********************************************************************  00250000
//*  STEP 1:  DROP ANY EXISTING STORED PROCEDURE CALLED DSN.DSN8EP2     00260000
//********************************************************************  00270000
//PH06SS01 EXEC PGM=IKJEFT01,DYNAMNBR=20                                00280000
//SYSTSPRT DD  SYSOUT=*                                                 00290000
//SYSTSIN  DD  *                                                        00300000
 DSN SYSTEM(DSN)                                                        00310000
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  00320000
      LIB('DSN!!0.RUNLIB.LOAD')      -                                  00330000
      PARM('RC0')                                                       00340000
//SYSPRINT DD  SYSOUT=*                                                 00350000
//SYSUDUMP DD  SYSOUT=*                                                 00360000
//SYSIN    DD *                                                         00370000
 SET CURRENT SQLID = 'SYSADM';                                          00371000
                                                                        00380000
 DROP PROCEDURE DSN8.DSN8EP2 RESTRICT;                                  00390000
                                                                        00400000
//*                                                                     00410000
//********************************************************************  00420000
//*  STEP 2:  CREATE SAMPLE STORED PROCEDURE DSN8.DSN8EP2               00430000
//********************************************************************  00440000
//PH06SS02 EXEC PGM=IKJEFT01,DYNAMNBR=20                                00450000
//SYSTSPRT DD  SYSOUT=*                                                 00460000
//SYSTSIN  DD  *                                                        00470000
 DSN SYSTEM(DSN)                                                        00480000
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  00490000
      LIB('DSN!!0.RUNLIB.LOAD')                                         00500000
//SYSPRINT DD  SYSOUT=*                                                 00510000
//SYSUDUMP DD  SYSOUT=*                                                 00520000
//SYSIN    DD *                                                         00530000
 SET CURRENT SQLID = 'SYSADM';                                          00531000
                                                                        00532000
 CREATE PROCEDURE                                                       00540000
   DSN8.DSN8EP2(                                                        00550000
     IN VARCHAR(4096)    CCSID EBCDIC,                                  00560000
    OUT INTEGER,                                                        00570000
    OUT INTEGER,                                                        00580000
    OUT INTEGER,                                                        00590000
    OUT VARCHAR(8320)    CCSID EBCDIC )                                 00600000
   LANGUAGE PLI                                                         00610000
   DETERMINISTIC                                                        00620000
   NO SQL                                                               00630000
   EXTERNAL NAME DSN8EP2                                                00640000
   PARAMETER STYLE GENERAL WITH NULLS                                   00650000
   COLLID DSN8EP!!                                                      00660000
   WLM ENVIRONMENT WLMENV                                               00670000
   ASUTIME LIMIT 5                                                      00680000
   STAY RESIDENT NO                                                     00690000
   PROGRAM TYPE MAIN                                                    00700000
   SECURITY DB2                                                         00710000
   NO DBINFO                                                            00720000
   RESULT SET 0                                                         00730000
   COMMIT ON RETURN NO;                                                 00740000
//*                                                                     00750000
//********************************************************************  00760000
//*  STEP 3:  PRE-COMPILE, COMPILE, AND LINK-EDIT THE STORED PROCEDURE  00770000
//********************************************************************  00780000
//PH06SS03  EXEC DSNHPLI,MEM=DSN8EP2,COND=(4,LT),                       00790000
//         PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)'          00800000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8EP2),                         00810000
//             DISP=SHR                                                 00820000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EP2),                     00830000
//             DISP=SHR                                                 00840000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00850000
//             DISP=SHR                                                 00860000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EP2),                      00870000
//             DISP=SHR                                                 00880000
//LKED.SYSIN DD  *                                                      00890000
 INCLUDE SYSLIB(DSNRLI)                                                 00900000
```

```
 INCLUDE SYSLIB(DSNTIAR)                                                00910000
//*                                                                     00920000
//********************************************************************* 00930000
//*  STEP 4:  BIND THE STORED PROCEDURE PACKAGE                         00940000
//*           NOTE: THIS STEP IS COMMENTED OUT FOR THE STORED           00950000
//*                 PROCEDURE SAMPLE APPLICATION BECAUSE IT CONTAINS    00960000
//*                 NO SQL STATEMENTS. IF YOUR STORED PROCEDURE         00970000
//*                 CONTAINS SQL STATEMENTS, YOU MUST BIND IT AS        00980000
//*                 A PACKAGE.                                          00990000
//********************************************************************* 01000000
//*PH06SS04   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                 01010000
//*DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                01020000
//*             DISP=SHR                                                01030000
//*SYSTSPRT DD SYSOUT=*                                                 01040000
//*SYSTSIN  DD *                                                        01050000
//* DSN SYSTEM(DSN)                                                     01060000
//* BIND PACKAGE(DSN8EP!!) MEMBER(DSN8EP2) -                            01070000
//*      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)             01080000
//*                                                                     01090000
```

Related reference

“Sample applications in TSO” on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6D

THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION PROGRAM, DSN8ED1, THAT DEMONSTRATES HOW TO CALL A Db2 STORED PROCEDURE THAT RETURNS A RESULT SET.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ6D                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 6                                         00050000
//*                     SAMPLE CALLER: STORED PROCEDURE WITH RESULT SET 00060000
//*                     C LANGUAGE                                      00070000
//*                                                                     00080000
//*                                                                     00090000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00100000
//*    5605-DB2                                                         00110000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00120000
//*                                                                     00130000
//*    STATUS = VERSION 11                                              00140000
//*                                                                     00150000
//*  FUNCTION = THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION     00160000
//*             PROGRAM, DSN8ED1, THAT DEMONSTRATES HOW TO CALL A DB2   00170000
//*             STORED PROCEDURE THAT RETURNS A RESULT SET.             00180000
//*                                                                     00190000
//*             DSN8ED1 ACCEPTS A DB2 COMMAND FROM STANDARD INPUT       00200000
//*             (SYSIN) AND PASSES IT AS A PARAMETER TO THE STORED      00210000
//*             PROCEDURE WHICH RUNS ON A REMOTE DB2 SUBSYSTEM (SEE     00220000
//*             DSNTEJ6T FOR DETAILS).  THE STORED PROCEDURE PLACES THE 00230000
//*             RESPONSES IN A RESULT SET AND DSN8ED1 EXTRACTS THEM AND 00240000
//*             PRINTS THEM TO STANDARD OUTPUT (SYSPRINT).              00250000
//*                                                                     00260000
//*  DEPENDENCIES:                                                      00270000
//*  (1) RUN SAMPLE JOB DSNTEJ6T AT THE SERVER SITE BEFORE RUNNING THIS 00280000
//*      JOB; DSNTEJ6T PREPARES THE SAMPLE STORED PROC W/ RESULT SET    00290000
//*  (2) RUN THIS JOB AT THE CLIENT SITE                                00300000
//*                                                                     00310000
//*                                                                     00320000
//* CHANGE ACTIVITY =                                                   00330000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00331000
//*               ADD SET CURRENT SQLID                                 00331101
//*                                                                     00332000
//********************************************************************* 00340000
//JOBLIB   DD DSN=DSN!!0.SDSNEXIT,DISP=SHR                              00350000
//         DD DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00360000
//         DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                           00370000
//         DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                           00380000
//*                                                                     00390000
//********************************************************************* 00400000
//*  STEP 1:  PRE-COMPILE, COMPILE, AND LINK-EDIT THE CALLING PROGRAM   00410000
//********************************************************************* 00420000
//PH06DS01 EXEC DSNHC,MEM=DSN8ED1,                                      00430000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00440000
//             SOURCE,XREF),                                            00450000
//         PARM.C='SOURCE LIST MAR(1,72) NORENT OPTFILE(DD:CCOPTS)',    00460000
```

```
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'                   00470000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED1),                    00480000
//             DISP=SHR                                                00490000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                              00500000
//             DISP=SHR                                                00510000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED1),                        00520000
//             DISP=SHR                                                00530000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED1),                     00540000
//             DISP=SHR                                                00550000
//LKED.SYSIN   DD *                                                    00560000
 INCLUDE SYSLIB(DSNELI)                                                00570000
 INCLUDE SYSLIB(DSNTIAR)                                               00580000
//*                                                                    00590000
//********************************************************************  00600000
//*  STEP 2:  BIND THE CALLING PROGRAM PACKAGE                          00610000
//********************************************************************  00620000
//PH06DS02   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                 00630000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                00640000
//             DISP=SHR                                                00650000
//SYSTSPRT DD SYSOUT=*                                                 00660000
//SYSPRINT DD SYSOUT=*                                                 00670000
//SYSUDUMP DD SYSOUT=*                                                 00680000
//SYSIN    DD *                                                        00690000
 SET CURRENT SQLID = 'SYSADM';                                         00691001
 GRANT BIND, EXECUTE ON PLAN DSN8ED1                                   00700002
   TO PUBLIC;                                                          00701002
//SYSTSIN  DD  *                                                       00710000
 DSN SYSTEM(DSN)                                                       00720000
 BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED1) -                              00730000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)               00740000
 BIND PACKAGE(SAMPLOC.DSN8ED!!) -                                      00750000
   MEMBER(DSN8ED1) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)   00760000
 BIND PLAN(DSN8ED1) -                                                  00770000
 PKLIST(DSN8ED!!.DSN8ED1, -                                            00780000
        SAMPLOC.DSN8ED!!.DSN8ED1 -                                     00790000
        SAMPLOC.DSN8ED!!.DSN8ED2) -                                    00800000
   ACTION(REPLACE) RETAIN +                                            00801000
   ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                           00810000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                               00820000
     LIB('DSN!!0.RUNLIB.LOAD')                                         00830000
//*                                                                    00840000
//********************************************************************  00850000
//*  STEP 3:  EXECUTE THE STORED PROCEDURE                              00860000
//********************************************************************  00870000
//PH06DS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                   00880000
//SYSTSPRT DD  SYSOUT=*                                                00890000
//SYSPRINT DD  SYSOUT=*                                                00900000
//SYSUDUMP DD  SYSOUT=*                                                00910000
//SYSTSIN  DD  *                                                       00920000
 DSN SYSTEM(DSN)                                                       00930000
 RUN PROGRAM(DSN8ED1) PLAN(DSN8ED1) PARMS('/SAMPLOC')                  00940000
 END                                                                   00950000
//SYSIN    DD *                                                        00960000
  -DISPLAY ARCHIVE;                                                    00970000
  -DISPLAY THREAD(*) DETAIL;                                           00980000
//*                                                                    00990000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6T

THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION PROGRAM, DSN8ED2, THAT DEMONSTRATES A Db2 STORED PROCEDURE THAT RETURNS A RESULT SET.

```
//********************************************************************  00010000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00020000
//*                     PHASE 6                                        00030000
//*                     SAMPLE STORED PROCEDURE WITH RESULT SET        00040000
//*                     C LANGUAGE                                     00050000
//*                                                                    00060000
//*                                                                    00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                            00080000
//*    5605-DB2                                                        00090000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.        00100000
//*                                                                    00110000
//*    STATUS = VERSION 10                                             00120000
```

```
//*
//*  FUNCTION = THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION
//*             PROGRAM, DSN8ED2, THAT DEMONSTRATES A DB2 STORED
//*             PROCEDURE THAT RETURNS A RESULT SET.
//*
//*             DSN8ED2 ACCEPTS A DB2 COMMAND PASSED AS AN INPUT
//*             PARAMETER FROM A CLIENT PROGRAM ON A REMOTE DB2
//*             SUBSYSTEM.  IT CALLS THE IFI UTILITY TO PROCESS THE
//*             COMMAND AND PLACES THE RESPONSES IN A TEMPORARY DB2
//*             TABLE SO THEY CAN BE RETURNED AS A RESULT SET TO THE
//*             CLIENT.
//*
//*  DEPENDENCIES:
//*  (1) RUN THIS JOB AT THE SERVER SITE BEFORE RUNNING SAMPLE JOB
//*      DSNTEJ6D AT THE CLIENT SITE
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 ADD SET CURRENT SQLID           DN1651_INST1 / DN1651
//*
//*********************************************************************
//JOBLIB   DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
//         DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//         DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//         DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//*
//*********************************************************************
//*  STEP 1:  DROP OBJECTS CREATED BY ANY PREVIOUS RUN OF DSNTEJ6T:
//*           - SAMPLE STORED PROCEDURE DSN8.DSN8ED2
//*           - GLOBAL TEMPORARY TABLE DSN8.DSN8ED2_RS_TBL
//*********************************************************************
//PH06TS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD')      -
      PARM('RC0')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 DROP PROCEDURE DSN8.DSN8ED2 RESTRICT;
 COMMIT;

 DROP TABLE DSN8.DSN8ED2_RS_TBL;
 COMMIT;

//*
//*********************************************************************
//*  STEP 2:  CREATE SAMPLE STORED PROCEDURE DSN8.DSN8ED2
//*           AND GLOBAL TEMPORARY TABLE DSN8.DSN8ED2_RS_TBL
//*********************************************************************
//PH06TS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,
//         COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 CREATE PROCEDURE
   DSN8.DSN8ED2(
     IN VARCHAR(4096)    CCSID EBCDIC,
    OUT INTEGER,
    OUT INTEGER,
    OUT INTEGER,
    OUT VARCHAR(880)     CCSID EBCDIC )
   LANGUAGE C
   DETERMINISTIC
   MODIFIES SQL DATA
   EXTERNAL NAME DSN8ED2
   PARAMETER STYLE GENERAL WITH NULLS
   COLLID DSN8ED!!
   WLM ENVIRONMENT WLMENV
   ASUTIME LIMIT 50
   STAY RESIDENT NO
   PROGRAM TYPE MAIN
```

```
   SECURITY DB2                                                         00900000
   NO DBINFO                                                            00910000
   RESULT SET 1                                                         00920000
   COMMIT ON RETURN NO;                                                 00930000
                                                                        00940000
 CREATE GLOBAL TEMPORARY TABLE DSN8.DSN8ED2_RS_TBL                      00950000
      ( RS_SEQUENCE   INTEGER      NOT NULL,                            00960000
        RS_DATA       CHAR( 80 )   NOT NULL )                           00970000
   CCSID EBCDIC;                                                        00980000
                                                                        00990000
//*                                                                     01000000
//*******************************************************************  01010000
//*  STEP 3:  PRE-COMPILE, COMPILE, AND LINK-EDIT THE STORED PROCEDURE  01020000
//*******************************************************************  01030000
//PH06TS03 EXEC DSNHC,MEM=DSN8ED2,                                      01040000
//         COND=(4,LT),                                                 01050000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     01060000
//             SOURCE,XREF),                                            01070000
//         PARM.C='SOURCE LIST MARGINS(1,72),RENT OPTFILE(DD:CCOPTS)',  01080000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT'                      01090000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED2),                     01100000
//             DISP=SHR                                                 01110000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01120000
//             DISP=SHR                                                 01130000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED2),                         01140000
//             DISP=SHR                                                 01150000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED2),                      01160000
//             DISP=SHR                                                 01170000
//LKED.SYSIN   DD *                                                     01180000
 INCLUDE SYSLIB(DSNRLI)                                                 01190000
 INCLUDE SYSLIB(DSNTIAR)                                                01200000
//*                                                                     01210000
//*******************************************************************  01220000
//*  STEP 4:  BIND THE STORED PROCEDURE PACKAGE                         01230000
//*******************************************************************  01240000
//PH06TS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01250000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,                                 01260000
//             DISP=SHR                                                 01270000
//SYSTSPRT DD  SYSOUT=*                                                 01280000
//SYSPRINT DD  SYSOUT=*                                                 01290000
//SYSUDUMP DD  SYSOUT=*                                                 01300000
//SYSTSIN  DD  *                                                        01310000
 DSN SYSTEM(DSN)                                                        01320000
 BIND PACKAGE(DSN8ED!!) -                                               01330000
      MEMBER(DSN8ED2) ACT(REP) -                                        01340000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01350000
//*                                                                     01360000
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSNTEJ61

This JCL creates a sample application, DSN8EC1, that demonstrates a Db2 stored procedure for IMS ODBA.

```
//*******************************************************************  00010000
//*  Name = DSNTEJ61                                                    00020000
//*                                                                     00030000
//*  Descriptive Name = DB2 Sample Application                          00040000
//*                     Phase 6                                         00050000
//*                     Sample Stored Procedure for IMS ODBA            00060000
//*                     Cobol Language                                  00070000
//*                                                                     00080000
//*                                                                     00090000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00100000
//*    5635-DB2                                                         00110000
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.         00120000
//*                                                                     00130000
//*    Status = VERSION 9                                               00140000
//*                                                                     00150000
//*  Function = This JCL creates a sample application, DSN8EC1, that    00160000
//*             demonstrates a DB2 stored procedure for IMS ODBA.       00170000
//*                                                                     00180000
//*             DSN8EC1 can be used to insert, retrieve, update,        00190000
//*             and delete rows in the IMS IVP telephone directory      00200000
//*             database, DFSIVD1.                                      00210000
```

```
//*
//*             DSN8EC1 has one input-only parm, five input/output
//*             parms, and three output-only parms.
//*             - Input only:
//*               (1) TDBCTLID  : ID of IMS subsystem where data resides
//*             - Input/Output:
//*               (2) COMMAND   : Action to be taken, or action taken
//*                   - ADD: Add an entry
//*                   - DEL: Delete an entry
//*                   - DIS: Retrieve an entry
//*                   - UPD: Update an entry
//*               (3) LAST_NAME : Operand for, or result of, COMMAND
//*               (4) FIRST_NAME:    "     "   "    "    "      "
//*               (5) EXTENSION :    "     "   "    "    "      "
//*               (6) ZIP-CODE  :    "     "   "    "    "      "
//*             - Output only:
//*               (7) AIBRETRN  : Return code from IMS AIB
//*               (8) AIBREASN  : Reason code from IMS AIB
//*               (9) ERROR-CALL: DL/I command executed
//*
//*
//*  Dependencies:
//*  (1) Run this job at the server site before running sample job
//*      DSNTEJ62 at the client site
//*  (2) The server site must have an IMS subsystem running IMS/ESA V6
//*      or a subsequent release
//*  (3) This IMS subsystem must have the following IMS IVP parts
//*      available
//*      (A) DFSIVD1, the IMS IVP telephone directory database
//*      (B) DFSIVP64, the IMS IVP Cobol PSB for BMP access to DFSIVD1
//*  (4) Specify the id for this IMS subsystem in DB2 sample job
//*      DSNTEJ62, step PH062S03
//*  (5) The server site must also have a WLM environment started by
//*      a proc that references the IMS reslib in both the STEPLIB DD
//*      and the DFSRESLB DD.  See the DB2 Installation Guide for more
//*      information.
//*  (6) Before running this job, verify that this WLM environment is
//*      the one specified in the CREATE PROCEDURE statement in step
//*      PH061S01.
//*
//*  Change Activity =
//*    11/07/2012 Add SET CURRENT SQLID           dn1651_inst1 / dn1651
//*
//*********************************************************************
//JOBLIB   DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
//         DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//         DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//         DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//*
//*********************************************************************
//*  STEP 1:  Drop the sample ODBA stored procedure, DSN8.DSN8EC1
//*********************************************************************
//PH061S01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD') -
      PARM('RC0')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 DROP PROCEDURE DSN8.DSN8EC1 RESTRICT;

//*
//*********************************************************************
//*  STEP 2:  Create the sample ODBA stored procedure, DSN8.DSN8EC1
//*********************************************************************
//PH061S02 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 CREATE PROCEDURE
```

```
   DSN8.DSN8EC1(
     IN CHAR(8)     CCSID EBCDIC,
  INOUT CHAR(8)     CCSID EBCDIC,
  INOUT CHAR(10)    CCSID EBCDIC,
  INOUT CHAR(10)    CCSID EBCDIC,
  INOUT CHAR(10)    CCSID EBCDIC,
  INOUT CHAR(7)     CCSID EBCDIC,
    OUT INT,
    OUT INT,
    OUT CHAR(4)     CCSID EBCDIC )
   FENCED
   RESULT SETS 0
   EXTERNAL NAME DSN8EC1
   LANGUAGE COBOL
   PARAMETER STYLE GENERAL
   NOT DETERMINISTIC
   NO SQL
   NO DBINFO
   NO COLLID
   WLM ENVIRONMENT WLMENV
   ASUTIME LIMIT 50
   STAY RESIDENT NO
   PROGRAM TYPE MAIN
   SECURITY DB2
   RUN OPTIONS 'TRAP(OFF),RPTOPTS(OFF),TERMTHDAC((QUIET),NONOVR)'
   COMMIT ON RETURN NO;
//*
//*********************************************************************
//*  Step 3:  Pre-compile, compile, and link-edit the stored procedure
//*********************************************************************
//PH061S03 EXEC DSNHICOB,MEM=DSN8EC1,
//         COND=(4,LT),
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
//         NOXREF,'SQL(DB2)','DEC(31)'),
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EC1),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8EC1),
//             DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EC1),
//             DISP=SHR
//LKED.SYSIN   DD *
 INCLUDE SYSLIB(DSNRLI)
 NAME DSN8EC1(R)
//*
//*********************************************************************
//*  Step 4:  Bind the stored procedure package
//*           Note: This step is commented out for sample stored
//*                 procedure DSN8EC1 because it contains no SQL
//*                 statements.  If your stored procedure contains
//*                 SQL statements, you must bind it as a package.
//*********************************************************************
//**PH061S04   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//**DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,
//**             DISP=SHR
//**SYSTSPRT DD SYSOUT=*
//**SYSPRINT DD SYSOUT=*
//**SYSUDUMP DD SYSOUT=*
//**SYSTSIN  DD *
//** DSN SYSTEM(DSN)
//** BIND PACKAGE(DSN8EC!!) MEMBER(DSN8EC1) -
//**      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ62

This JCL prepares and executes a sample application program, DSN8EC2, that demonstrates how to call a Db2 stored procedure for IMS ODBA.

```
//*********************************************************************
//*  Name = DSNTEJ62
//*
```

```
//*  Descriptive Name = DB2 Sample Application                          00040000
//*                     Phase 6                                         00050000
//*                     Sample Client: Stored procedure for IMS ODBA    00060000
//*                     Cobol Language                                  00070000
//*                                                                     00080000
//*                                                                     00090000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00100000
//*    5635-DB2                                                         00110000
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.         00120000
//*                                                                     00130000
//*    STATUS = VERSION 11                                              00140001
//*                                                                     00150000
//*  Function = This JCL prepares and executes a sample application     00160000
//*             program, DSN8EC2, that demonstrates how to call a DB2   00170000
//*             stored procedure for IMS ODBA.  The results are         00180000
//*             directed to the SYSOUT DD.                              00190000
//*                                                                     00200000
//*             DSN8EC2 accepts a runtime parameter in step PH062S03    00210000
//*             that specifies -both- the DB2 server location name      00220000
//*             where the stored procedure is registered -and- the id   00230000
//*             of the IMS subsystem where the ODBA activity is to      00240000
//*             occur.  You must modify this job to provide the IMS     00250000
//*             subsystem id.  See step PH062S03 for details.           00260000
//*                                                                     00270000
//*  Dependencies:                                                      00280000
//*  (1) Run sample job DSNTEJ61 at the server site before running this 00290000
//*      job; DSNTEJ61 prepares the sample stored proc for IMS ODBA     00300000
//*  (2) Modify this job as directed in step PH062S03                   00310000
//*  (3) Run this job at the client site                                00320000
//*                                                                     00330000
//*                                                                     00340000
//*  Change activity =                                                  00350000
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00351000
//*               Add SET CURRENT SQLID                                 00351103
//*                                                                     00352000
//*********************************************************************  00360000
//JOBLIB   DD DSN=DSN!!0.SDSNEXIT,DISP=SHR                              00370000
//         DD DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00380000
//         DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                           00390000
//         DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                           00400000
//*                                                                     00410000
//*********************************************************************  00420000
//*  Step 1:  Pre-compile, compile, and link-edit the client program    00430000
//*********************************************************************  00440000
//PH062S01 EXEC DSNHICOB,MEM=DSN8EC2,                                   00450000
//         COND=(4,LT),                                                 00460000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00470000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00480000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),       00490000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP'                           00500000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EC2),                     00510000
//             DISP=SHR                                                 00520000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00530000
//             DISP=SHR                                                 00540000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8EC2),                         00550000
//             DISP=SHR                                                 00560000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EC2),                      00570000
//             DISP=SHR                                                 00580000
//LKED.SYSIN   DD *                                                     00590000
 INCLUDE SYSLIB(DSNELI)                                                 00600000
 INCLUDE SYSLIB(DSNTIAR)                                                00610000
//*                                                                     00620000
//*********************************************************************  00630000
//*  Step 2:  Bind the client program package and plan                  00640000
//*********************************************************************  00650000
//PH062S02   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                  00660000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                 00670000
//             DISP=SHR                                                 00680000
//SYSTSPRT DD SYSOUT=*                                                  00690000
//SYSPRINT DD SYSOUT=*                                                  00700000
//SYSUDUMP DD SYSOUT=*                                                  00710000
//SYSTSIN  DD  *                                                        00720000
 DSN SYSTEM(DSN)                                                        00730000
 BIND PACKAGE(DSN8EC!!) MEMBER(DSN8EC2) -                               00740000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00750000
 BIND PACKAGE(SAMPLOC.DSN8EC!!) -                                       00760000
      MEMBER(DSN8EC2) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)00770000
 BIND PLAN(DSN8EC2) -                                                   00780000
      PKLIST(DSN8EC!!.DSN8EC2, -                                        00790000
             SAMPLOC.DSN8EC!!.DSN8EC2) -                                00800000
      ACTION(REPLACE) RETAIN +                                          00801000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         00810000
```

```
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
     LIB('DSN!!0.RUNLIB.LOAD')
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8EC2
   TO PUBLIC;
//*
//*********************************************************************
//*  STEP 3:  Invoke the client for the IMS ODBA stored procedure
//*    Note:  The PARMS keyword in the RUN statement below accepts a
//*           single argument that specifies the DB2 server location
//*           name -and- the IMS subsystem id, in that order and
//*           separated by a single blank character.
//*
//*           Example: PARMS('SANTA_TERESA_LAB IMSP')
//*
//*           Verify that the PARMS keyword below specifies the name
//*           of the DB2 server location name where you ran DSNTEJ61.
//*
//*           Change the string ?IMSID? to the id of the IMS subsystem
//*           where you want the ODBA-directed activity to occur.  This
//*           subsystem must reside on the same server as the DB2
//*           server and must be running IMS/ESA V6 or a subsequent
//*           release.
//*
//*********************************************************************
//PH062S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
   RUN PROGRAM(DSN8EC2) -
   PLAN(DSN8EC2)        -
   PARMS('SAMPLOC ?IMSID?')
 END
//*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ63

This JCL prepares DSN8ES1, a sample SQL procedure that accepts a department number and returns salary and bonus data for employees in that department from the Db2 sample data base.

```
//**********************************************************************
//*  Name = DSNTEJ63
//*
//*  Descriptive Name =
//*    DB2 Sample Application
//*    Phase 6
//*    Sample SQL Procedure
//*    SQL Procedure Language
//*
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 10
//*
//*  Function =
//*    This JCL prepares DSN8ES1, a sample SQL procedure that
//*    accepts a department number and returns salary and bonus data
//*    for employees in that department from the DB2 sample data base.
//*
//*  Pseudocode =
//*    PH063S01 Step     Drop objects created by prior runs of this job
//*    PH063S02 Step     Prepare DSN8ES1 load module from DSN8ES1 src
//*    PH063S03 Step     Bind DSN8ES1 package in collection DSN8ES!!
//*                      Register the stored procedure using generated
//*                      DDL from the precompiler
//*                      Create the global temporary table required by
//*                      the result set
//*
```

```
//*  Dependencies =                                                     00310000
//*  (1) This job requires the DB2-provided JCL procedure DSNHSQL       00320000
//*  (2) Run this job prior to running the client job DSNTEJ64          00330000
//*                                                                     00340000
//*  Notes =                                                            00350000
//*                                                                     00360000
//*  Change Activity =                                                  00370000
//*    11/07/2012 Add SET CURRENT SQLID           dn1651_inst1 / dn1651 00370100
//*                                                                     00371000
//********************************************************************* 00380000
//*                                                                     00390000
//JOBLIB   DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR                             00400000
//         DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                             00410000
//         DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                          00420000
//         DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                          00430000
//*                                                                     00440000
//*                                                                     00450000
//********************************************************************* 00460000
//*  STEP 1:  Drop any pre-existing entries for stored proc DSN8ES1     00470000
//*           and the global temporary table for its result set         00480000
//********************************************************************* 00490000
//PH063S01 EXEC PGM=IKJEFT01,DYNAMNBR=20                                00500000
//SYSTSPRT DD  SYSOUT=*                                                 00510000
//SYSTSIN  DD  *                                                        00520000
 DSN SYSTEM(DSN)                                                        00530000
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  00540000
      LIB('DSN!!0.RUNLIB.LOAD') -                                       00550000
      PARM('RC0')                                                       00560000
//SYSPRINT DD  SYSOUT=*                                                 00570000
//SYSUDUMP DD  SYSOUT=*                                                 00580000
//SYSIN    DD *                                                         00590000
  SET CURRENT SQLID = 'SYSADM';                                         00591000
                                                                        00600000
  DROP PROCEDURE DSN8.DSN8ES1 RESTRICT;                                 00610000
  COMMIT;                                                               00620000
                                                                        00630000
  DROP TABLE DSN8.DSN8ES1_RS_TBL;                                       00640000
  COMMIT;                                                               00650000
                                                                        00660000
//*                                                                     00670000
//********************************************************************* 00680000
//*  Step 2:  Pre-compile, compile, and link-edit the stored procedure  00690000
//********************************************************************* 00700000
//PH063S02 EXEC DSNHSQL,MEM=DSN8ES1,                                    00710000
//         COND=(4,LT),                                                 00720000
//         PARM.PC=('HOST(SQL),SOURCE,XREF,MAR(1,72),CCSID(37)',        00730000
//            'STDSQL(NO)'),                                            00740000
//         PARM.PCC=('HOST(C),SOURCE,XREF,MAR(1,80),CCSID(37)',         00750000
//            'TWOPASS,STDSQL(NO)'),                                    00760000
//         PARM.C='SOURCE LIST MARGINS(1,80) NOSEQ LO RENT              00770000
//             LOCALE("SAA") OPTFILE(DD:CCOPTS)',                       00780000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT'                      00790000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00800000
//             DISP=SHR                                                 00810000
//PC.SYSIN     DD DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES1),                     00820000
//             DISP=SHR                                                 00830000
//PC.SYSUT2    DD DSN=&&SPDDL,DISP=(,PASS),                             00840000
//             UNIT=SYSDA,SPACE=(TRK,1),                                00850000
//             DCB=(RECFM=FB,LRECL=80)                                  00860000
//PCC.DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ES1),                     00870000
//             DISP=SHR                                                 00880000
//PCC.SYSLIB   DD DSN=DSN!!0.SRCLIB.DATA,                               00890000
//             DISP=SHR                                                 00900000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ES1),                      00910000
//             DISP=SHR                                                 00920000
//LKED.SYSIN   DD *                                                     00930000
 INCLUDE SYSLIB(DSNRLI)                                                 00940000
 NAME DSN8ES1(R)                                                        00950000
//*                                                                     00960000
//********************************************************************* 00970000
//*  STEP 3:  Create the global temp table for DSN8ES1's result set     00980000
//*           Register DSN8ES1 in SYSIBM.SYSROUTINES                    00990000
//*           Bind the package for DSN8ES1                              01000000
//********************************************************************* 01010000
//PH063S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,                               01020000
//             COND=(4,LT)                                              01030000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA(DSN8ES1),                        01040000
//             DISP=SHR                                                 01050000
//SYSTSPRT DD  SYSOUT=*                                                 01060000
//SYSTSIN  DD  *                                                        01070000
 DSN SYSTEM(DSN)                                                        01080000
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                 01090000
```

```
       LIB('DSN!!0.RUNLIB.LOAD') -
       PARM('SQLTERM(%)')
  BIND PACKAGE(DSN8ES!!) -
       QUALIFIER(DSN8!!0) -
       MEMBER(DSN8ES1) -
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 END
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
  SET CURRENT SQLID = 'SYSADM'
  %
  CREATE GLOBAL TEMPORARY TABLE
         DSN8.DSN8ES1_RS_TBL
         ( RS_SEQUENCE   INTEGER      NOT NULL,
           RS_EMPNO      CHAR(6)      NOT NULL,
           RS_FIRSTNME   CHAR(12)     NOT NULL,
           RS_LASTNAME   CHAR(15)     NOT NULL,
           RS_SALARY     DECIMAL(9, 2) NOT NULL,
           RS_BONUS      DECIMAL(9, 2) NOT NULL )
         CCSID EBCDIC
  %
//         DD  DSN=&&SPDDL,DISP=(OLD,DELETE)   <- From preceding step
//*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ64

This JCL prepares and executes DSN8ED3, a sample caller for the sample SQL procedure DSN8ES1.

```
//*********************************************************************
//*  Name = DSNTEJ64
//*
//*  Descriptive Name =
//*    DB2 Sample Application
//*    Phase 6
//*    Sample Caller for sample SQL Procedure DSN8ES1
//*    C Language
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    Status = VERSION 11
//*
//*  Function =
//*    This JCL prepares and executes DSN8ED3, a sample caller for the
//*    sample SQL procedure DSN8ES1.  DSN8ED3 passes a sample
//*    department number to DSN8ES1, then processes what is returned:
//*    - Parameters containing:
//*      - The total earnings (salaries and bonuses) of employees in
//*        that department
//*      - The number of employees who got a bonus
//*    - A result set containing a row of data (serial no, first and
//*      last name, salary, and bonus) for each employee who got a
//*      bonus
//*
//*  Pseudocode =
//*    PH064S01 Step     Prepare DSN8ED3 load module from DSN8ED3 src
//*    PH064S02 Step     Bind DSN8ED3 package in collection DSN8ED!!
//*                      Bind DSN8ED3 plan from DSN8ED!! and DSN8ES!!
//*                        collection ids
//*    PH064S03 Step     Run DSN8ED3 to call stored procedure DSN8ES1
//*
//*  Dependencies =
//*  (1) Run sample job DSNTEJ63 prior to running this job
//*  (2) This job requires the DB2-provided JCL procedure DSNHC
//*
//*  Notes =
//*
//*  Change Activity =
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651
//*               Add SET CURRENT SQLID
//*
//*********************************************************************
```

```
//JOBLIB   DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR                             00430000
//         DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                             00440000
//         DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                          00450000
//         DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                          00460000
//*                                                                     00470000
//********************************************************************* 00480000
//*  Step 1:  Pre-compile, compile, and link-edit DSN8ED3               00490000
//********************************************************************* 00500000
//PH064S01 EXEC DSNHC,MEM=DSN8ED3,                                      00510000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00520000
//             SOURCE,XREF),                                            00530000
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',   00540000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'                    00550000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED3),                     00560000
//             DISP=SHR                                                 00570000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00580000
//             DISP=SHR                                                 00590000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED3),                         00600000
//             DISP=SHR                                                 00610000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED3),                      00620000
//             DISP=SHR                                                 00630000
//LKED.SYSIN   DD *                                                     00640000
 INCLUDE SYSLIB(DSNELI)                                                 00650000
 INCLUDE SYSLIB(DSNTIAR)                                                00660000
//*                                                                     00670000
//********************************************************************* 00680000
//*  Step 2:  Bind DSN8ED3's PLAN from its package and DSN8ES1's pkg    00690000
//********************************************************************* 00700000
//PH064S02   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                  00710000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                 00720000
//             DISP=SHR                                                 00730000
//SYSTSPRT DD SYSOUT=*                                                  00740000
//SYSPRINT DD SYSOUT=*                                                  00750000
//SYSUDUMP DD SYSOUT=*                                                  00760000
//SYSIN    DD *                                                         00770000
 SET CURRENT SQLID = 'SYSADM';                                          00771001
 GRANT BIND, EXECUTE ON PLAN DSN8ED3                                    00780002
   TO PUBLIC;                                                           00781002
//SYSTSIN  DD  *                                                        00790000
 DSN SYSTEM(DSN)                                                        00800000
 BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED3) -                               00810000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00820000
 BIND PACKAGE(SAMPLOC.DSN8ED!!) -                                       00830000
      MEMBER(DSN8ED3) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)00840000
 BIND PLAN(DSN8ED3) -                                                   00850000
      PKLIST(DSN8ED!!.DSN8ED3, -                                        00860000
             SAMPLOC.DSN8ED!!.DSN8ED3, -                                00870000
             SAMPLOC.DSN8ES!!.DSN8ES1) -                                00880000
      ACTION(REPLACE) RETAIN +                                          00881000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         00890000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                 00900000
     LIB('DSN!!0.RUNLIB.LOAD')                                          00910000
//*                                                                     00920000
//********************************************************************* 00930000
//*  STEP 3:  Get Bonus and Salary report for department D11            00940000
//********************************************************************* 00950000
//PH064S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00960000
//SYSTSPRT DD  SYSOUT=*                                                 00970000
//SYSPRINT DD  SYSOUT=*                                                 00980000
//SYSUDUMP DD  SYSOUT=*                                                 00990000
//SYSTSIN  DD  *                                                        01000000
 DSN SYSTEM(DSN)                                                        01010000
  RUN PROGRAM(DSN8ED3) PLAN(DSN8ED3) PARMS('/D11 SAMPLOC')              01020000
 END                                                                    01030000
//*                                                                     01040000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ65

This JCL does the following.

```
//********************************************************************* 00010000
//*  Name = DSNTEJ65                                                    00020000
//*                                                                     00030000
//*  Descriptive Name =                                                 00040000
```

```
//*    DB2 Sample Application                                           00050000
//*    Phase 6                                                          00060000
//*    - Sample C Caller for DB2 SQL Procedures Processor (DSNTPSMP)    00070000
//*    - Sample SQL Procedure for DSNTPSMP to prepare                   00080000
//*    - Sample C Caller for SQL Procedure prepared by DSNTPSMP         00090000
//*                                                                     00100000
//*                                                                     00110000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00120000
//*    5605-DB2                                                         00130000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00140000
//*                                                                     00150000
//*    STATUS = VERSION 11                                              00160000
//*                                                                     00170000
//*  Function =                                                         00180000
//*    This JCL does the following:                                     00190000
//*    (1) Prepares and binds DSN8ED4, a sample caller for DSNTPSMP,    00200000
//*        the DB2 Stored Procedures Processor.                         00210000
//*    (2) Invokes DSN8ED4 to prequalify that the server has DSNTPSMP   00220000
//*        at the proper interface level supported by the this client   00230000
//*    (3) Invokes DSN8ED4 to pass a sample SQL Procedure, DSN8.DSN8ES2,00240000
//*        to DSNTPSMP for preparation                                  00250000
//*    (4) Prepares, binds, and executes DSN8ED5, a sample caller for   00260000
//*        DSN8.DSN8ES2                                                 00270000
//*                                                                     00280000
//*  Pseudocode =                                                       00290000
//*    PH065S01 Step   Prepare DSN8ED4 (sample caller of DSNTPSMP)      00300000
//*    PH065S02 Step   Bind DSN8ED4                                     00310000
//*    PH065S03 Step   Call DSN8ED4 to request DSNTPSMP QUERYLEVEL      00320000
//*    PH065S04 Step   Call DSN8ED4 to pass DSN8.DSN8ES2 to DSNTPSMP    00330000
//*    PH065S05 Step   Prepare DSN8ED5 (sample caller of DSN8.DSN8ES2)  00340000
//*    PH065S06 Step   Bind DSN8ED5                                     00350000
//*    PH065S07 Step   Call DSN8ED5 to call DSN8.DSN8ES2                00360000
//*                                                                     00370000
//*  Dependencies =                                                     00380000
//*    (1) Sample program requires DSNTPSMP (the DB2 SQL Procedures     00390000
//*        Processor)                                                   00400000
//*                                                                     00410000
//*  Notes =                                                            00420000
//*                                                                     00430000
//*  Change Activity =                                                  00440000
//*  PQ43444 04/12/2001 Remove LEOPTS DD from PH065S03                  00450000
//*  PQ56601 03/14/2002 Add reminder not to use continuation characters 00460000
//*                     in BIND options input to DSN8ED4                00470000
//*  D55199  12/08/2003 Add new step 3 to prequalify DSNTPSMP level     00480000
//*  dn1651  11/07/2012 Add RETAIN to BIND PLAN stmts      dn1651_inst1 00481000
//*                     Add SET CURRENT SQLID                           00482001
//*                                                                     00483000
//********************************************************************* 00490000
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT                             00500000
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00510000
//         DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN                          00520000
//         DD  DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                          00530000
//*                                                                     00540000
//********************************************************************* 00550000
//*  Step 1:  Prepare DSN8ED4, caller for DSNTPSMP                      00560000
//********************************************************************* 00570000
//PH065S01 EXEC DSNHC,MEM=DSN8ED4,                                      00580000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00590000
//             SOURCE,XREF),                                            00600000
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',   00610000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'                    00620000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED4),                     00630000
//             DISP=SHR                                                 00640000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00650000
//             DISP=SHR                                                 00660000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED4),                         00670000
//             DISP=SHR                                                 00680000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED4),                      00690000
//             DISP=SHR                                                 00700000
//LKED.SYSIN   DD *                                                     00710000
  INCLUDE SYSLIB(DSNELI)                                                00720000
  INCLUDE SYSLIB(DSNTIAR)                                               00730000
//*                                                                     00740000
//********************************************************************* 00750000
//*  Step 2:  Bind DSN8ED4's PLAN                                       00760000
//********************************************************************* 00770000
//PH065S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00780000
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                         00790000
//SYSTSPRT DD  SYSOUT=*                                                 00800000
//SYSPRINT DD  SYSOUT=*                                                 00810000
//SYSUDUMP DD  SYSOUT=*                                                 00820000
//SYSIN    DD  *                                                        00830000
```

```
  SET CURRENT SQLID = 'SYSADM';                                         00831001
  GRANT BIND, EXECUTE ON PLAN DSN8ED4                                   00840002
    TO PUBLIC;                                                          00841002
//SYSTSIN  DD  *                                                        00850000
  DSN SYSTEM(DSN)                                                       00860000
    BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED4) -                            00870000
         ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)             00880000
    BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED4) -                    00890000
                 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)     00900000
    BIND PLAN(DSN8ED4) -                                                00910000
         PKLIST(DSN8ED!!.DSN8ED4, -                                     00920000
                SAMPLOC.DSN8ED!!.DSN8ED4, -                             00930000
                SAMPLOC.DSNREXCS.DSNREXX) -                             00940000
         ACTION(REPLACE) RETAIN +                                       00941000
         ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                      00950000
    RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                               00960000
        LIB('DSN!!0.RUNLIB.LOAD')                                       00970000
  END                                                                   00980000
//*                                                                     00990000
//********************************************************************* 01000000
//*  STEP 3: Invoke DSN8ED4 to pass a QUERYLEVEL request to DSNTPSMP.   01010000
//*          This is a prequalification that the server has a DSNTPSMP  01020000
//*          at the correct Interface level for our DSN8ED4 client.     01030000
//*          Parms: (* used as place holders)                           01040000
//*          (1) QUERYLEVEL                                             01050000
//*          (2) *                                                      01060000
//*          (3) *                                                      01070000
//*          (4) (optional) name of server where DSNTPSMP is to be run  01080000
//*          Note: DSN8ED4 requires all the same definitions be present 01090000
//*                as on a BUILD request, even though only the function 01100000
//*                request QUERYLEVEL is passed to DSNTPSMP.            01110000
//********************************************************************* 01120000
//PH065S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01130000
//SYSTSPRT DD  SYSOUT=*                                                 01140000
//SYSPRINT DD  SYSOUT=*                                                 01150000
//SYSUDUMP DD  SYSOUT=*                                                 01160000
//SYSTSIN  DD  *                                                        01170000
  DSN SYSTEM(DSN)                                                       01180000
    RUN PROGRAM(DSN8ED4) PLAN(DSN8ED4) -                                01190000
    PARMS('QUERYLEVEL * * SAMPLOC')                                     01200000
  END                                                                   01210000
//PCOPTS   DD  *                                                        01220000
//COPTS    DD  *                                                        01230000
//PLKDOPTS DD  *                                                        01240000
//LKEDOPTS DD  *                                                        01250000
//BINDOPTS DD  *                                                        01260000
//SQLIN    DD  *                                                        01270000
//*                                                                     01280000
//REPORT01 DD  SYSOUT=*,DCB=(RECFM=FBA)                                 01290000
//REPORT02 DD  SYSOUT=*                                                 01300000
//REPORT03 DD  SYSOUT=*                                                 01310000
//REPORT04 DD  SYSOUT=*                                                 01320000
//REPORT05 DD  SYSOUT=*                                                 01330000
//REPORT06 DD  SYSOUT=*                                                 01340000
//*                                                                     01350000
//********************************************************************* 01360000
//*  STEP 4:  Invoke DSN8ED4 to pass sample SQL Procedure DSN8.DSN8ES2  01370000
//*           to DSNTPSMP                                               01380000
//*           Parms:                                                    01390000
//*           (1) operation to be performed by DSNTPSMP                 01400000
//*           (2) schema.name of SQL proc to be prepared by DSNTPSMP    01410000
//*           (3) SQL authid to be used when calling DSNTPSMP           01420000
//*           (4) (optional) name of server where DSNTPSMP is to be run 01430000
//*           Note: Options passed in the PCOPTS, COPTS, PLKDOPTS, and  01440000
//*                 BINDOPTS DDs can span more than one input record.   01450000
//*                 Do not use continuation characters (+ or -) to      01460000
//*                 continue BIND options onto the next BINDOPTS record 01470000
//********************************************************************* 01480000
//PH065S04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01490000
//SYSTSPRT DD  SYSOUT=*                                                 01500000
//SYSPRINT DD  SYSOUT=*                                                 01510000
//SYSUDUMP DD  SYSOUT=*                                                 01520000
//SYSTSIN  DD  *                                                        01530000
  DSN SYSTEM(DSN)                                                       01540000
    RUN PROGRAM(DSN8ED4) PLAN(DSN8ED4) -                                01550000
    PARMS('REBUILD DSN8.DSN8ES2 AUTHID SAMPLOC')                        01560000
  END                                                                   01570000
//PCOPTS   DD  *                                                        01580000
  SOURCE,XREF,MAR(1,80),STDSQL(NO)                                      01590000
//COPTS    DD  *                                                        01600000
  SOURCE LIST MAR(1,80) NOSEQ LO RENT                                   01610000
//PLKDOPTS DD  *                                                        01620000
```

```
//LKEDOPTS DD  *                                                        01630000
  AMODE=31,RMODE=ANY,MAP,RENT                                           01640000
//BINDOPTS DD  *                                                        01650000
  PACKAGE(DSN8ES!!)                                                     01660000
  QUALIFIER(DSN8!!0) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) 01670000
//SQLIN    DD  DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES2),                        01680000
//             DISP=SHR                                                 01690000
//*                                                                     01700000
//REPORT01 DD  SYSOUT=*,DCB=(RECFM=FBA)                                 01710000
//REPORT02 DD  SYSOUT=*                                                 01720000
//REPORT03 DD  SYSOUT=*                                                 01730000
//REPORT04 DD  SYSOUT=*                                                 01740000
//REPORT05 DD  SYSOUT=*                                                 01750000
//REPORT06 DD  SYSOUT=*                                                 01760000
//*                                                                     01770000
//********************************************************************** 01780000
//*  Step 5:  Prepare DSN8ED5, sample caller of DSN8.DSN8ES2            01790000
//********************************************************************** 01800000
//PH065S05 EXEC DSNHC,MEM=DSN8ED5,COND=(4,LT),                          01810000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     01820000
//             SOURCE,XREF),                                            01830000
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',   01840000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'                    01850000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED5),                     01860000
//             DISP=SHR                                                 01870000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01880000
//             DISP=SHR                                                 01890000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED5),                         01900000
//             DISP=SHR                                                 01910000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED5),                      01920000
//             DISP=SHR                                                 01930000
//LKED.SYSIN   DD *                                                     01940000
 INCLUDE SYSLIB(DSNELI)                                                 01950000
 INCLUDE SYSLIB(DSNTIAR)                                                01960000
//*                                                                     01970000
//********************************************************************** 01980000
//*  Step 6:  Bind DSN8ED5's PLAN                                       01990000
//********************************************************************** 02000000
//PH065S06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02010000
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                         02020000
//SYSTSPRT DD  SYSOUT=*                                                 02030000
//SYSPRINT DD  SYSOUT=*                                                 02040000
//SYSUDUMP DD  SYSOUT=*                                                 02050000
//SYSIN    DD  *                                                        02060000
  SET CURRENT SQLID = 'SYSADM';                                         02061001
  GRANT BIND, EXECUTE ON PLAN DSN8ED5                                   02070002
    TO PUBLIC;                                                          02071002
//SYSTSIN  DD  *                                                        02080000
  DSN SYSTEM(DSN)                                                       02090000
    BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED5) -                            02100000
         ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)             02110000
    BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED5) -                    02120000
                 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)     02130000
    BIND PLAN(DSN8ED5) -                                                02140000
         PKLIST(DSN8ED!!.DSN8ED5, -                                     02150000
                SAMPLOC.DSN8ED!!.DSN8ED5, -                             02160000
                SAMPLOC.DSN8ES!!.*) -                                   02170000
         ACTION(REPLACE) RETAIN +                                       02171000
         ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                      02180000
    RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                               02190000
        LIB('DSN!!0.RUNLIB.LOAD')                                       02200000
  END                                                                   02210000
//*                                                                     02220000
//********************************************************************** 02230000
//*  STEP 7:  Invoke DSN8ED5 to call sample SQL Procedure DSN8.DSN8ES2  02240000
//********************************************************************** 02250000
//PH065S07 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02260000
//SYSTSPRT DD  SYSOUT=*                                                 02270000
//SYSPRINT DD  SYSOUT=*                                                 02280000
//SYSUDUMP DD  SYSOUT=*                                                 02290000
//SYSTSIN  DD  *                                                        02300000
  DSN SYSTEM(DSN)                                                       02310000
    RUN PROGRAM(DSN8ED5) PLAN(DSN8ED5) -                                02320000
    PARMS('1500.00 SAMPLOC')                                            02330000
  END                                                                   02340000
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSNTEJ6W

This JCL does the following.

```
//*                                                                     00010000
//*    DB2 Sample Application                                           00020000
//*    Phase 6                                                          00030000
//*    Sample caller for Stored Procedure WLM_REFRESH                   00040000
//*    IBM C/C++ for z/OS                                               00050000
//*                                                                     00060000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00070000
//*    5605-DB2                                                         00080000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00090000
//*                                                                     00100000
//*  Status = VERSION 11                                                00110000
//*                                                                     00120000
//*  Function =                                                         00130000
//*    This JCL does the following:                                     00140000
//*    * Prepares and executes DSN8ED6, a sample caller of the          00150000
//*      WLM_REFRESH stored procedure.  WLM_REFRESH refreshes a         00160000
//*      WLM environment specified as an input parameter using an       00170000
//*      authorization ID also specified as an input parameter.  The    00180000
//*      authorization ID must have READ access on a resource profile   00190000
//*      called !DSN!.WLM_REFRESH.!WLMENV!                              00200000
//*      Use job DSNTIJRA job step DSNTWR to create and permit access   00220000
//*      to this resource profile.                                      00230000
//*    * Optional: Prepares DSNTWR, the external module for             00240000
//*      WLM_REFRESH.  The DSNTWR module is provided in SDSNLOAD so     00250000
//*      preparing it is required only if you maintain a customized     00260000
//*      copy of DSNTWRS, the sample source code for DSNTWR.            00270000
//*                                                                     00280000
//*  Pseudocode =                                                       00290000
//*    PH06WS01 Step     Optional: Prepare DSNTWR, the external module  00330000
//*                      for SYSPROC.WLM_REFRESH                        00340000
//*                      -> Uncomment and run this step if you want to  00350000
//*                         override the DB2-supplied DSNTWR module     00360000
//*    PH06WS02 Step     Optional: Bind the package for DSNTWR          00370000
//*                      -> Uncomment and run this step only if you     00380000
//*                         also uncomment and run the step PH06WS01    00390000
//*    PH06WS03 Step     Prepare DSN8ED6                                00400000
//*    PH06WS04 Step     Bind the plan and package for DSN8ED6          00410000
//*    PH06WS05 Step     Invoke DSN8ED6                                 00420000
//*                                                                     00430000
//*  Dependencies =                                                     00440000
//*  (1) This job requires the DB2-provided JCL procedures DSNHC        00450000
//*  (2) Run this job only after running jobs DSNTIJTM and DSNTIJSG     00460000
//*  (3) The DSN8ED6 program receives parameters that contain the name  00470000
//*      of the WLM environment to be refreshed and the authorization   00480000
//*      ID to be used for the request.  The authorization ID must have 00490000
//*      READ access an a resource profile called                       00491000
//*        !DSN!.WLM_REFRESH.!WLMENV!                                   00492000
//*      Use job DSNTIJRA job step DSNTWR to create and permit access   00493000
//*      to this resource profile.                                      00494000
//*                                                                     00500000
//*  Notes =                                                            00510000
//*                                                                     00520000
//*  Change Activity =                                                  00530000
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00531000
//*               Add SET CURRENT SQLID                                 00531102
//*                                                                     00532000
//********************************************************************* 00540000
//*                                                                     00550000
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT                             00560000
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00570000
//         DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN                          00580000
//*                                                                     00590000
//*  //*                                                                00720000
//*  //*   Step  1 (Optional): Prepare DSNTWR, the external module for  00730000
//*  //*                       WLM_REFRESH                              00740000
//*  //*                                                                00750000
//*  //PH06WS01 EXEC DSNHASM,COND=(4,LT),                               00760000
//*  //            MEM=DSNTWR,                                          00770000
//*  //            PARM.PC='HOST(ASM),STDSQL(NO)',                      00780000
//*  //            PARM.ASM='RENT,OBJECT,NODECK',                       00790000
//*  //            PARM.LKED='LIST,LET,XREF,REUS,                       00800000
//*  //            AMODE=31,RMODE=ANY,RENT'                             00810000
//*  //PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSNTWR),                 00820000
```

```
//*  //             DISP=SHR                                            00830000
//*  //PC.SYSLIB    DD DSN=DSN!!0.SDSNSAMP,                             00840000
//*  //             DISP=SHR                                            00850000
//*  //PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSNTWRS),                    00860000
//*  //             DISP=SHR                                            00870000
//*  //ASM.SYSLIB   DD DSN=SYS1.MACLIB,                                 00880000
//*  //             DISP=SHR                                            00890000
//*  //             DD DSN=CEE.V!R!M!.SCEEMAC,                          00900000
//*  //             DISP=SHR                                            00910000
//*  //             DD DSN=DSN!!0.SDSNMACS,                             00920000
//*  //             DISP=SHR                                            00930000
//*  //             DD DSN=DSN!!0.SDSNSAMP,                             00940000
//*  //             DISP=SHR                                            00950000
//*  //LKED.SYSLMOD DD DSN=DSN!!0.SDSNEXIT(DSNTWR),                     00960000
//*  //             DISP=SHR                                            00970000
//*  //LKED.SYSLIB  DD DSN=CEE.V!R!M!.SCEELKED,                         00980000
//*  //             DISP=SHR                                            00990000
//*  //             DD DSN=CEE.V!R!M!.SCEERUN,                          01000000
//*  //             DISP=SHR                                            01010000
//*  //             DD DSN=CEE.V!R!M!.SCEESPC,                          01020000
//*  //             DISP=SHR                                            01030000
//*  //             DD DSN=CEE.V!R!M!.SCEESPCO,                         01040000
//*  //             DISP=SHR                                            01050000
//*  //             DD DSN=DSN!!0.SDSNLOAD,                             01060000
//*  //             DISP=SHR                                            01070000
//*  //LKED.SYSIN   DD *                                                01080000
//*   INCLUDE SYSLIB(DSNRLI)                                            01090000
//*   SETCODE AC(1)                                                     01100000
//*   NAME DSNTWR(R)                                                    01110000
//*  /*                                                                 01120000
//*  //*                                                                01130000
//*  //*   Step  2 (Optional): Bind the package for DSNTWR              01140000
//*  //*                                                                01150000
//*  //PH06WS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)               01160000
//*  //DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                    01170000
//*  //SYSTSPRT DD  SYSOUT=*                                            01180000
//*  //SYSPRINT DD  SYSOUT=*                                            01190000
//*  //SYSUDUMP DD  SYSOUT=*                                            01200000
//*  //SYSTSIN  DD  *                                                   01210000
//*    DSN SYSTEM(DSN)                                                  01220000
//*      BIND PACKAGE(DSNTWR) MEMBER(DSNTWR) -                          01230000
//*           ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)        01240000
//*    END                                                              01250000
//*  /*                                                                 01260000
//*                                                                     01270000
//*   Step  3: Prepare DSN8ED6, sample caller of WLM_REFRESH            01280000
//*                                                                     01290000
//PH06WS03 EXEC DSNHC,MEM=DSN8ED6,COND=(4,LT),                          01300000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     01310000
//             SOURCE,XREF),                                            01320000
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',   01330000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT,REUS'                 01340000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED6),                     01350000
//             DISP=SHR                                                 01360000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01370000
//             DISP=SHR                                                 01380000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED6),                         01390000
//             DISP=SHR                                                 01400000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED6),                      01410000
//             DISP=SHR                                                 01420000
//LKED.SYSIN   DD *                                                     01430000
 INCLUDE SYSLIB(DSNELI)                                                 01440000
 INCLUDE SYSLIB(DSNTIAR)                                                01450000
//*                                                                     01460000
//*   Step  4: Bind the package and plan for DSN8ED6                    01470000
//*                                                                     01480000
//PH06WS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01490000
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                         01500000
//SYSTSPRT DD  SYSOUT=*                                                 01510000
//SYSPRINT DD  SYSOUT=*                                                 01520000
//SYSUDUMP DD  SYSOUT=*                                                 01530000
//SYSTSIN  DD  *                                                        01540000
  DSN SYSTEM(DSN)                                                       01550000
    BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED6) -                            01560000
         ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)             01570000
    BIND PLAN(DSN8ED6) -                                                01580000
         PKLIST(DSN8ED!!.DSN8ED6, -                                     01590000
                DSNTWR.DSNTWR) -                                        01600000
         ACTION(REPLACE) RETAIN +                                       01601000
         ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                      01610000
    RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!)  -                              01620000
        LIB('DSN!!0.RUNLIB.LOAD')                                       01630000
```

```
  END
//SYSIN    DD  *
  SET CURRENT SQLID = 'SYSADM';
  GRANT BIND, EXECUTE ON PLAN DSN8ED6
    TO PUBLIC;
//*
//*   Step  5: Invoke DSN8ED6
//*
//PH06WS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DSN)
    RUN PROGRAM(DSN8ED6) PLAN(DSN8ED6) -
        LIB('DSN!!0.RUNLIB.LOAD') -
        PARMS('!WLMENV! !DSN! !ID!')
  END
/*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ6Z

This JCL prepares and executes DSN8ED7, a sample caller of ADMIN_INFO_SYSPARM, a Db2-provided stored procedure that returns the current settings of your Db2 subsystem parameters.

```
//*
//*    DB2 Sample Application
//*    Phase 6
//*    Sample Caller of Stored Procedure - ADMIN_INFO_SYSPARM
//*    C language
//*
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*  Status = VERSION 11
//*
//*  Function =
//*    This JCL prepares and executes DSN8ED7, a sample caller of
//*    ADMIN_INFO_SYSPARM, a DB2-provided stored procedure that returns
//*    the current settings of your DB2 subsystem parameters.  After
//*    calling ADMIN_INFO_SYSPARM, DSN8ED7 formats the results in a
//*    report format.
//*
//*  Pseudocode =
//*    PH06ZS01 Step     Prepare DSN8ED7
//*    PH06ZS02 Step     Bind the plan and package for DSN8ED7
//*    PH06ZS03 Step     Invoke DSN8ED7
//*
//*  Dependencies =
//*   - This job requires the DB2-provided JCL procedure DSNHC
//*
//*  Notes =
//*
//*  Change Activity =
//*    11/07/2012 Convert from SYSPROC.DSNWZP     dn1651_inst1 / dn1651
//*                 to SYSPROC.ADMIN_INFO_SYSPARM
//*               Add RETAIN to BIND PLAN stmts
//*               Add SET CURRENT SQLID
//*
//*********************************************************************
//*
//JOBLIB   DD  DISP=SHR,DSN=DSN!!0.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//         DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//*   Step  1: Prepare DSN8ED7, sample caller of ADMIN_INFO_SYSPARM
//*
//PH06ZS01 EXEC DSNHC,MEM=DSN8ED7,COND=(4,LT),
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//             SOURCE,XREF),
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT,REUS'
```

```
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED7),                     00440000
//             DISP=SHR                                                 00450000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00460000
//             DISP=SHR                                                 00470000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED7),                         00480000
//             DISP=SHR                                                 00490000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED7),                      00500000
//             DISP=SHR                                                 00510000
//LKED.SYSIN   DD *                                                     00520000
 INCLUDE SYSLIB(DSNELI)                                                 00530000
 INCLUDE SYSLIB(DSNTIAR)                                                00540000
//*                                                                     00550000
//*   Step  2: Bind the package and plan for DSN8ED7                    00560000
//*                                                                     00570000
//PH06ZS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00580000
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                         00590000
//SYSTSPRT DD  SYSOUT=*                                                 00600000
//SYSPRINT DD  SYSOUT=*                                                 00610000
//SYSUDUMP DD  SYSOUT=*                                                 00620000
//SYSTSIN  DD  *                                                        00630000
  DSN SYSTEM(DSN)                                                       00640000
    BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED7) -                            00650000
         ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)             00660000
    BIND PLAN(DSN8ED7) -                                                00670000
         PKLIST(DSN8ED!!.DSN8ED7 ) -                                    00680000
         ACTION(REPLACE) RETAIN +                                       00681002
         ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                      00690002
    RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                               00700000
        LIB('DSN!!0.RUNLIB.LOAD')                                       00710000
  END                                                                   00720000
//SYSIN    DD  *                                                        00730000
  SET CURRENT SQLID = 'SYSADM';                                         00731003
  GRANT BIND, EXECUTE ON PLAN DSN8ED7                                   00740005
    TO PUBLIC;                                                          00741005
//*                                                                     00750000
//*   Step  3: Invoke DSN8ED7                                           00760000
//*                                                                     00770000
//PH06ZS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00780000
//SYSTSPRT DD  SYSOUT=*                                                 00790000
//SYSPRINT DD  SYSOUT=*                                                 00800000
//SYSUDUMP DD  SYSOUT=*                                                 00810000
//SYSTSIN  DD  *                                                        00820000
  DSN SYSTEM(DSN)                                                       00830000
    RUN PROGRAM(DSN8ED7) PLAN(DSN8ED7) -                                00840000
        LIB('DSN!!0.RUNLIB.LOAD')                                       00850000
  END                                                                   00860000
/*                                                                      00870000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ66

This JCL does the following.

```
//********************************************************************* 00010000
//*  Name = DSNTEJ66                                                    00020000
//*                                                                     00030000
//*  Descriptive Name = DB2 Sample Application - Native SQL Procedure   00040000
//*                     Phase 6                                         00050000
//*                                                                     00060000
//*                                                                     00070000
//*    Licensed Materials - Property of IBM                             00080000
//*    5605-DB2                                                         00090000
//*    (C) COPYRIGHT 2006, 2010 IBM Corp.  All Rights Reserved.         00100000
//*                                                                     00110000
//*    STATUS = Version 11                                              00120000
//*                                                                     00130000
//*  Function = This JCL does the following:                            00140000
//*             - Creates a sample native SQL procedure called          00150000
//*               DSN8.DSN8ES3 that generates and returns (by result    00160000
//*               set) a CREATE PROCEDURE statement for a given stored  00170000
//*               procedure.                                            00180000
//*             - Prepares and executes a sample caller of DSN8ES3      00190000
//*               called DSN8ED9.                                       00200000
//*             - Shows how to use ALTER PROCEDURE ... ADD VERSION to   00210000
//*               create a version V2 of DSN8ES3 that does the same     00220000
```

```
//*                thing as the original version but also adds a
//*                terminating semicolon at the end of the generated
//*                CREATE PROCEDURE statement
//*              - Shows how to ALTER ACTIVATE version V2 to make it
//*                the active version of DSN8ES3
//*              - Shows how to DEPLOY DSN8ES3 at a remote site
//*
//*  Restrictions =
//*    As part of the setup to DEPLOY DSN8ES3, the DSNTEP2 application
//*    needs to be able to connect to the remote site.
//*
//*  Notice =
//*
//*  Pseudocode =
//*    PH066S01 Step     Drop objects created by prior runs of this job
//*    PH066S02 Step     Create the global temporary table for
//*                      DSN8.DSN8ES3
//*    PH066S03 Step     Prepare DSN8ES3 as a native SQL procedure
//*                      -> Also generates a package called
//*                         DSN8.DSN8ES3
//*    PH066S04 Step     Prepare DSN8ED9, sample caller for the DSN8ES3
//*                      SQL proc
//*    PH066S05 Step     Bind the plan for DSN8ED9
//*    PH066S06 Step     Execute DSN8ED9 to request a CREATE PROC
//*                      statement for the stored procedure
//*                      SYSPROC.DSNUTILS
//*    PH066S07 Step     Create a work copy of the DSN8ES3 source code
//*    PH066S08 Step     Use TSO edit to modify the work copy into an
//*                      ALTER PROCEDURE that will make a trivial
//*                      change to DSN8ES3 as VERSION V2
//*                      -> The generated CREATE PROC statement will be
//*                         terminated by a semicolon
//*    PH066S09 Step     Save the work copy as DSN8ES3 in
//*                      DSN!!0.NEW.SDSNSAMP
//*    PH066S10 Step     Process the ALTER PROCEDURE DSN8ES3 to ADD
//*                      VERSION V2
//*                      -> Also generates a package called
//*                         DSN8.DSN8ES3 (VERSION V2)
//*    PH066S11 Step     Activate V2 as the current version of DSN8ES3
//*    PH066S12 Step     Execute DSN8ED9 to request a CREATE PROC
//*                      statement for SYSPROC.DSNUTILU
//*                      -> When using DSN8ES3 V2, it's terminated by a
//*                      semicolon
//*    PH066S13 Step     Setup to DEPLOY DSN8ES3: Create a global
//*                      temporary table on the remote server
//*                      -> To rerun this step, uncomment the DROP
//*                         and COMMIT statements
//*    PH066S14 Step     DEPLOY DSN8ES3 on the remote server
//*    PH066S15 Step     Bind the plan for DSN8ED9 on the remote server
//*    PH066S16 Step     Execute DSN8ED9 to request a CREATE PROC
//*                      statement for SYSPROC.DSNUTILS at the remote
//*                      site
//*
//*  Change Activity =
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651
//*               Add SET CURRENT SQLID
//*
//*********************************************************************
//*
//JOBLIB    DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
//          DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//          DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//*
//* Step  1: Drop objects created by prior runs of this job
//*
//PH066S01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) +
     PLAN(DSNTIA!!) +
     LIB('DSN!!0.RUNLIB.LOAD') +
     PARM('RC0')
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSIN     DD *
  SET CURRENT SQLID = 'SYSADM';
  DROP PROCEDURE DSN8.DSN8ES3;
  COMMIT;
  DROP TABLE DSN8.DSN8ES3_RS_TBL;
  COMMIT;
//WORKCOPY  DD DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY,
```

```
//             DISP=(MOD,DELETE),                                       01010000
//             UNIT=SYSDA,SPACE=(TRK,0)                                 01020000
//*                                                                     01030000
//* Step  2: Create the global temporary table for DSN8.DSN8ES3         01040000
//*                                                                     01050000
//PH066S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01060000
//SYSTSPRT  DD SYSOUT=*                                                 01070000
//SYSTSIN   DD *                                                        01080000
 DSN SYSTEM(DSN)                                                        01090000
 RUN PROGRAM(DSNTIAD) +                                                 01100000
     PLAN(DSNTIA!!) +                                                   01110000
     LIB('DSN!!0.RUNLIB.LOAD')                                          01120000
//SYSPRINT  DD SYSOUT=*                                                 01130000
//SYSUDUMP  DD SYSOUT=*                                                 01140000
//SYSIN     DD *                                                        01150000
  SET CURRENT SQLID = 'SYSADM';                                         01151001
  CREATE GLOBAL TEMPORARY TABLE                                         01160000
         DSN8.DSN8ES3_RS_TBL                                            01170000
         ( RS_SEQUENCE   INTEGER      NOT NULL,                         01180000
           RS_LINE       CHAR(80)     NOT NULL )                        01190000
         CCSID EBCDIC                                                   01200000
//*                                                                     01210000
//* Step  3: Prepare DSN8ES3 as a native SQL procedure                  01220000
//*          -> Also generates a package called DSN8.DSN8ES3            01230000
//*                                                                     01240000
//PH066S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01250000
//SYSTSPRT  DD SYSOUT=*                                                 01260000
//SYSPRINT  DD SYSOUT=*                                                 01270000
//SYSUDUMP  DD SYSOUT=*                                                 01280000
//SYSTSIN   DD *                                                        01290000
 DSN SYSTEM(DSN)                                                        01300000
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +                                 01310000
      LIB('DSN!!0.RUNLIB.LOAD') PARMS('/SQLTERM(%)')                    01320000
 END                                                                    01330000
//SYSIN     DD *                                                        01340001
  SET CURRENT SQLID = 'SYSADM'                                          01350001
     %                                                                  01350101
//          DD DISP=SHR,                                                01351001
//             DSN=DSN!!0.SDSNSAMP(DSN8ES3)                             01352001
//          DD *                                                        01360000
     %                                                                  01370000
//*                                                                     01380000
//* Step  4: Prepare DSN8ED9, sample caller for the DSN8ES3 SQL proc    01390000
//*                                                                     01400000
//PH066S04 EXEC DSNHC,MEM=DSN8ED9,COND=(4,LT),                          01410000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     01420000
//             SOURCE,XREF),                                            01430000
//         PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',   01440000
//         PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'                    01450000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED9),                     01460000
//             DISP=SHR                                                 01470000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01480000
//             DISP=SHR                                                 01490000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8ED9),                         01500000
//             DISP=SHR                                                 01510000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED9),                      01520000
//             DISP=SHR                                                 01530000
//LKED.SYSIN   DD *                                                     01540000
 INCLUDE SYSLIB(DSNELI)                                                 01550000
 INCLUDE SYSLIB(DSNTIAR)                                                01560000
//*                                                                     01570000
//* Step  5: Bind the plan for DSN8ED9                                  01580000
//*                                                                     01590000
//PH066S05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01600000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                 01610000
//             DISP=SHR                                                 01620000
//SYSTSPRT  DD SYSOUT=*                                                 01630000
//SYSPRINT  DD SYSOUT=*                                                 01640000
//SYSUDUMP  DD SYSOUT=*                                                 01650000
//SYSIN     DD *                                                        01660000
 SET CURRENT SQLID = 'SYSADM';                                          01661001
 GRANT BIND, EXECUTE ON PLAN DSN8ED9                                    01670002
   TO PUBLIC;                                                           01671002
//SYSTSIN   DD *                                                        01680000
 DSN SYSTEM(DSN)                                                        01690000
 BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED9) +                               01700000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                01710000
 BIND PLAN(DSN8ED9) +                                                   01720000
      PKLIST(DSN8ED!!.DSN8ED9) +                                        01730000
      ACTION(REPLACE) RETAIN +                                          01731000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                         01740000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +                                 01750000
```

```
     LIB('DSN!!0.RUNLIB.LOAD')                                          01760000
//*                                                                     01770000
//* Step  6: Execute DSN8ED9 to request a CREATE PROC statement         01780000
//*          for the stored procedure named SYSPROC.DSNUTILS            01790000
//*                                                                     01800000
//PH066S06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01810000
//SYSTSPRT  DD SYSOUT=*                                                 01820000
//SYSPRINT  DD SYSOUT=*                                                 01830000
//SYSUDUMP  DD SYSOUT=*                                                 01840000
//SYSTSIN   DD *                                                        01850000
 DSN SYSTEM(DSN)                                                        01860000
  RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +                                  01870000
      LIB('DSN!!0.RUNLIB.LOAD') +                                       01880000
      PARMS('/SYSPROC DSNUTILS')                                        01890000
 END                                                                    01900000
//*                                                                     01910000
//* Step  7: Create a work copy of the DSN8ES3 source code              01920000
//*                                                                     01930000
//PH066S07 EXEC PGM=IEBGENER,COND=(4,LT)                                01940000
//SYSIN     DD DUMMY                                                    01950000
//SYSPRINT  DD SYSOUT=*                                                 01960000
//SYSUT1    DD DISP=SHR,                                                01970000
//             DSN=DSN!!0.SDSNSAMP(DSN8ES3)                             01980000
//SYSUT2    DD DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY,                        01990000
//             DISP=(,CATLG,DELETE),                                    02000000
//             UNIT=SYSDA,                                              02010000
//             SPACE=(TRK,1),                                           02020000
//             DCB=(RECFM=FB,LRECL=80)                                  02030000
//*                                                                     02040000
//* Step  8: Use TSO edit to modify the work copy into an ALTER PROCE-  02050000
//*          DURE that will make a trivial change to DSN8ES3 VERSION V2 02060000
//*          -> The generated CREATE PROC statement will now be         02070000
//*             terminated by a semicolon                               02080000
//*                                                                     02090000
//PH066S08 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02100000
//SYSTSPRT  DD SYSOUT=*                                                 02110000
//SYSPRINT  DD SYSOUT=*                                                 02120000
//SYSUDUMP  DD SYSOUT=*                                                 02130000
//SYSTSIN   DD *                                                        02140000
  EDIT 'DSN!!0.DSN8.DSN8ES3.WORKCOPY' +                                 02150000
        DATA OLD NONUM NORECOVER ASIS                                   02160000
  FIND /CREATE PROCEDURE DSN8.DSN8ES3/                                  02170000
  CHANGE * 1 /CREATE PROCEDURE/ALTER PROCEDURE /                        02180000
  INSERT    ADD VERSION V2                                              02190000
  FIND /PARAMETER CCSID EBCDIC/                                         02200000
  DELETE * 1                                                            02210000
  FIND /U100: -- Finish up/                                             02220000
  CHANGE * 1 /Finish up                /Add terminating semicolon/      02230000
  INSERT    SET LINE = ';';                                             02240000
  INSERT    SET RETURN_POINT = 'DONE';                                  02250000
  INSERT    GOTO INSERTLINE;                                            02260000
  LIST 1 9999                                                           02270000
  END SAVE                                                              02280000
//*                                                                     02290000
//* Step  9: Save in DSN!!0.NEW.SDSNSAMP                                02300000
//*                                                                     02310000
//PH066S09 EXEC PGM=IEBGENER,COND=(4,LT)                                02320000
//SYSIN     DD DUMMY                                                    02330000
//SYSPRINT  DD SYSOUT=*                                                 02340000
//SYSUT1    DD DISP=(OLD,DELETE),                                       02350000
//             DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY                         02360000
//SYSUT2    DD DISP=SHR,                                                02370000
//             DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES3)                         02380000
//*                                                                     02390000
//* Step 10: Process the ALTER PROCEDURE DSN8ES3 to ADD VERSION V2      02400000
//*          -> Also generates a package called DSN8.DSN8ES3 (V2)       02410000
//*                                                                     02420000
//PH066S10 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02430000
//SYSTSPRT  DD SYSOUT=*                                                 02440000
//SYSPRINT  DD SYSOUT=*                                                 02450000
//SYSUDUMP  DD SYSOUT=*                                                 02460000
//SYSTSIN   DD *                                                        02470000
 DSN SYSTEM(DSN)                                                        02480000
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +                                 02490000
      LIB('DSN!!0.RUNLIB.LOAD') PARMS('/SQLTERM(%)')                    02500000
 END                                                                    02510000
//SYSIN     DD *                                                        02510101
  SET CURRENT SQLID = 'SYSADM'                                          02511001
     %                                                                  02512001
//          DD DISP=SHR,                                                02520001
//             DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES3)                         02530000
//          DD *                                                        02540000
```

```
     %                                                                  02550000
//*                                                                     02560000
//* Step 11: Activate V2 as the current version of DSN8ES3              02570000
//*                                                                     02580000
//PH066S11 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02590000
//SYSTSPRT  DD SYSOUT=*                                                 02600000
//SYSPRINT  DD SYSOUT=*                                                 02610000
//SYSUDUMP  DD SYSOUT=*                                                 02620000
//SYSTSIN   DD *                                                        02630000
 DSN SYSTEM(DSN)                                                        02640000
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +                                 02650000
      LIB('DSN!!0.RUNLIB.LOAD')                                         02660000
 END                                                                    02670000
//SYSIN     DD *                                                        02680000
 SET CURRENT SQLID = 'SYSADM';                                          02681001
 ALTER PROCEDURE DSN8.DSN8ES3 ACTIVATE VERSION V2;                      02690000
//*                                                                     02700000
//* Step 12: Execute DSN8ED9 to request a CREATE PROC statement         02710000
//*          for SYSPROC.DSNUTILU                                       02720000
//*          -> When using DSN8ES3 V2, it's terminated by a semicolon   02730000
//*                                                                     02740000
//PH066S12 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02750000
//SYSTSPRT  DD SYSOUT=*                                                 02760000
//SYSPRINT  DD SYSOUT=*                                                 02770000
//SYSUDUMP  DD SYSOUT=*                                                 02780000
//SYSTSIN   DD *                                                        02790000
 DSN SYSTEM(DSN)                                                        02800000
  REBIND PACKAGE(DSN8ED!!.DSN8ED9) +                                    02801003
      PLANMGMT(OFF)                                                     02802003
  RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +                                  02810000
      LIB('DSN!!0.RUNLIB.LOAD') +                                       02820000
      PARMS('/SYSPROC DSNUTILU')                                        02830000
 END                                                                    02840000
//*                                                                     02850000
//* Step 13: Setup to DEPLOY DSN8ES3 - Create a global temporary        02860000
//*          table on the remote server                                 02870000
//*                                                                     02880000
//PH066S13 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02890000
//SYSTSPRT  DD SYSOUT=*                                                 02900000
//SYSPRINT  DD SYSOUT=*                                                 02910000
//SYSUDUMP  DD SYSOUT=*                                                 02920000
//SYSTSIN   DD *                                                        02930000
 DSN SYSTEM(DSN)                                                        02940000
 RUN PROGRAM(DSNTEP2) +                                                 02950000
     PLAN(DSNTEP!!) +                                                   02960000
      LIB('DSN!!0.RUNLIB.LOAD')                                         02970000
//SYSIN     DD *                                                        02980000
  SET CURRENT SQLID = 'SYSADM';                                         02981001
  CONNECT TO SAMPLOC;                                                   02990000
* DROP TABLE DSN8.DSN8ES3_RS_TBL;                                       03000000
* COMMIT;                                                               03010000
  CREATE GLOBAL TEMPORARY TABLE                                         03020000
         DSN8.DSN8ES3_RS_TBL                                            03030000
         ( RS_SEQUENCE   INTEGER      NOT NULL,                         03040000
           RS_LINE       CHAR(80)     NOT NULL )                        03050000
         CCSID EBCDIC;                                                  03060000
//*                                                                     03070000
//* Step 14: DEPLOY DSN8ES3 on the remote server                        03080000
//*                                                                     03090000
//PH066S14 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    03100000
//SYSTSPRT  DD SYSOUT=*                                                 03110000
//SYSPRINT  DD SYSOUT=*                                                 03120000
//SYSUDUMP  DD SYSOUT=*                                                 03130000
//SYSTSIN   DD *                                                        03140000
 DSN SYSTEM(DSN)                                                        03150000
 BIND PACKAGE(SAMPLOC.DSN8) +                                           03160000
      DEPLOY(DSN8.DSN8ES3) +                                            03170000
      COPYVER(V2) +                                                     03180000
      ACTION(REP)                                                       03190000
//*                                                                     03200000
//* Step 15: Bind the plan for DSN8ED9 on the remote server             03210000
//*                                                                     03220000
//PH066S15 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    03230000
//DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA,                                 03240000
//             DISP=SHR                                                 03250000
//SYSTSPRT  DD SYSOUT=*                                                 03260000
//SYSPRINT  DD SYSOUT=*                                                 03270000
//SYSUDUMP  DD SYSOUT=*                                                 03280000
//SYSIN     DD *                                                        03290000
 SET CURRENT SQLID = 'SYSADM';                                          03291001
 GRANT BIND, EXECUTE ON PLAN DSN8ED9                                    03300002
   TO PUBLIC;                                                           03301002
```

```
//SYSTSIN   DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED9) +
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 BIND PLAN(DSN8ED9) +
      PKLIST(DSN8ED!!.DSN8ED9, +
             SAMPLOC.DSN8ED!!.DSN8ED9, +
             SAMPLOC.DSN8ES!!.*) +
      ACTION(REPLACE) RETAIN +
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +
     LIB('DSN!!0.RUNLIB.LOAD')
//*
//* Step 16: Execute DSN8ED9 to request a CREATE PROC statement for
//*          SYSPROC.DSNUTILS at the remote site
//*
//PH066S16 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSTSIN   DD *
 DSN SYSTEM(DSN)
  RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +
      LIB('DSN!!0.RUNLIB.LOAD') +
      PARMS('/SYSPROC DSNUTILS SAMPLOC')
 END
//*
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ2U

THIS JCL PREPARES THE FOLLOWING Db2 USER-DEFINED FUNCTIONS (UDF'S) AND A DRIVER PROGRAM TO INVOKE THEM.

```
//*********************************************************************
//*  NAME = DSNTEJ2U
//*
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//*                     PHASE 2
//*                     USER DEFINED FUNCTIONS (C/C++)
//*
//*    Licensed Materials - Property of IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.
//*
//*    STATUS = Version 11
//*
//*  FUNCTION = THIS JCL PREPARES THE FOLLOWING DB2 USER-DEFINED
//*             FUNCTIONS (UDF'S) AND A DRIVER PROGRAM TO INVOKE THEM.
//*
//*     NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF
//*             THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION
//*
//*             THIS JOB IS RUN AFTER PHASE 1.
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651
//*               Add SET CURRENT SQLID
//*
//*********************************************************************
//*
//JOBLIB  DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR
//        DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//        DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//*
//*        STEP  1: DROP ANY EXISTING DB2 SAMPLE UDF'S
//*
//PH02US01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD)          -
     PLAN(DSNTIA!!)            -
     LIB('DSN!!0.RUNLIB.LOAD') -
```

```
      PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCDDVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCDVVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUADV     RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCTTVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCTVVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUATV     RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCYFV    RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUCYFVV   RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8EUDND     RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8EUDNV     RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8EUMND     RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8EUMNV     RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTINV    RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTINVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTINVVV  RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTISV    RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTISVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTISVVV  RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTILV    RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTILVV   RESTRICT;
 DROP SPECIFIC FUNCTION  DSN8.DSN8DUTILVVV  RESTRICT;

 DROP SPECIFIC FUNCTION  DSN8.DSN8DUWFV     RESTRICT;
//*
//*        STEP  2: DEFINE SAMPLE UDF'S TO DB2
//*
//PH02US02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD)           -
     PLAN(DSNTIA!!)             -
     LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID = 'SYSADM';

 CREATE FUNCTION
   DSN8.ALTDATE(
     VARCHAR(13) CCSID EBCDIC )
   RETURNS
     VARCHAR(17) CCSID EBCDIC
   SPECIFIC DSN8.DSN8DUADV
   LANGUAGE C
   DETERMINISTIC
   NO SQL
   EXTERNAL NAME DSN8DUAD
   PARAMETER STYLE DB2SQL
   NULL CALL
   NO EXTERNAL ACTION
   NO SCRATCHPAD
   NO FINAL CALL
   ALLOW PARALLEL
   NO COLLID
   ASUTIME LIMIT 10
   STAY RESIDENT NO
   PROGRAM TYPE SUB
   WLM ENVIRONMENT WLMENV
   SECURITY DB2
   NO DBINFO;

 CREATE FUNCTION
   DSN8.ALTDATE(
     VARCHAR(17) CCSID EBCDIC,
     VARCHAR(13) CCSID EBCDIC,
     VARCHAR(13) CCSID EBCDIC )
   RETURNS
     VARCHAR(17) CCSID EBCDIC
```

```
  SPECIFIC DSN8.DSN8DUCDVVV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8DUCD
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.ALTDATE(
    DATE,
    VARCHAR(13) CCSID EBCDIC,
    VARCHAR(13) CCSID EBCDIC )
  RETURNS
    VARCHAR(17) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUCDDVV
  SOURCE SPECIFIC DSN8.DSN8DUCDVVV;

CREATE FUNCTION
  DSN8.ALTTIME(
    VARCHAR(14) CCSID EBCDIC )
  RETURNS
    VARCHAR(11) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUATV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8DUAT
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.ALTTIME(
    VARCHAR(11) CCSID EBCDIC,
    VARCHAR(14) CCSID EBCDIC,
    VARCHAR(14) CCSID EBCDIC )
  RETURNS
    VARCHAR(11) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUCTVVV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8DUCT
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.ALTTIME(
    TIME,
```

```
    VARCHAR(14) CCSID EBCDIC,
    VARCHAR(14) CCSID EBCDIC )
  RETURNS
    VARCHAR(11) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUCTTVV
  SOURCE SPECIFIC DSN8.DSN8DUCTVVV;

CREATE FUNCTION
  DSN8.CURRENCY(
    FLOAT,
    VARCHAR(2)  CCSID EBCDIC )
  RETURNS
    VARCHAR(19) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUCYFV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8DUCY
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.CURRENCY(
    FLOAT,
    VARCHAR(2)  CCSID EBCDIC,
    VARCHAR(5)  CCSID EBCDIC )
  RETURNS
    VARCHAR(19) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUCYFVV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8DUCY
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.DAYNAME(
    VARCHAR(10) CCSID EBCDIC )
  RETURNS
    VARCHAR(9)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8EUDNV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8EUDN
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

```

```
CREATE FUNCTION
  DSN8.DAYNAME(
    DATE )
  RETURNS
    VARCHAR(9)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8EUDND
  SOURCE SPECIFIC DSN8.DSN8EUDNV;

CREATE FUNCTION
  DSN8.MONTHNAME(
    VARCHAR(10) CCSID EBCDIC )
  RETURNS
    VARCHAR(9)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8EUMNV
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME DSN8EUMN
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  NO COLLID
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE SUB
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.MONTHNAME(
    DATE )
  RETURNS
    VARCHAR(9)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8EUMND
  SOURCE SPECIFIC DSN8.DSN8EUMNV;

CREATE FUNCTION
  DSN8.TABLE_NAME(
    VARCHAR(18) CCSID EBCDIC )
  RETURNS
    VARCHAR(18) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTINV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_NAME(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC )
  RETURNS
    VARCHAR(18) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTINVV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
```

```
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_NAME(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC,
    VARCHAR(16) CCSID EBCDIC )
  RETURNS
    VARCHAR(18) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTINVVV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_SCHEMA(
    VARCHAR(18) CCSID EBCDIC )
  RETURNS
    VARCHAR(8)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTISV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_SCHEMA(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC )
  RETURNS
    VARCHAR(8)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTISVV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;
```

```
CREATE FUNCTION
  DSN8.TABLE_SCHEMA(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC,
    VARCHAR(16) CCSID EBCDIC )
  RETURNS
    VARCHAR(8)  CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTISVVV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_LOCATION(
    VARCHAR(18) CCSID EBCDIC )
  RETURNS
    VARCHAR(16) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTILV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_LOCATION(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC )
  RETURNS
    VARCHAR(16) CCSID EBCDIC
  SPECIFIC DSN8.DSN8DUTILVV
  LANGUAGE C
  DETERMINISTIC
  READS SQL DATA
  EXTERNAL NAME DSN8DUTI
  PARAMETER STYLE DB2SQL
  NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  COLLID DSN8DU!!
  ASUTIME LIMIT 10
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV
  SECURITY DB2
  NO DBINFO;

CREATE FUNCTION
  DSN8.TABLE_LOCATION(
    VARCHAR(18) CCSID EBCDIC,
    VARCHAR(8)  CCSID EBCDIC,
    VARCHAR(16) CCSID EBCDIC )
  RETURNS
    VARCHAR(16) CCSID EBCDIC
```

```
    SPECIFIC DSN8.DSN8DUTILVVV
    LANGUAGE C
    DETERMINISTIC
    READS SQL DATA
    EXTERNAL NAME DSN8DUTI
    PARAMETER STYLE DB2SQL
    NULL CALL
    NO EXTERNAL ACTION
    NO SCRATCHPAD
    NO FINAL CALL
    ALLOW PARALLEL
    COLLID DSN8DU!!
    ASUTIME LIMIT 10
    STAY RESIDENT NO
    PROGRAM TYPE MAIN
    WLM ENVIRONMENT WLMENV
    SECURITY DB2
    NO DBINFO;

  CREATE FUNCTION
    DSN8.WEATHER(
      VARCHAR(44) CCSID EBCDIC )
    RETURNS
      TABLE(
        CITY          VARCHAR(30) CCSID EBCDIC,
        TEMP_IN_F     INTEGER,
        HUMIDITY      INTEGER,
        WIND          VARCHAR(5)  CCSID EBCDIC,
        WIND_VELOCITY INTEGER,
        BAROMETER     FLOAT,
        FORECAST      VARCHAR(25) CCSID EBCDIC )
    SPECIFIC DSN8.DSN8DUWFV
    LANGUAGE C
    DETERMINISTIC
    NO SQL
    EXTERNAL NAME DSN8DUWF
    PARAMETER STYLE DB2SQL
    NULL CALL
    NO EXTERNAL ACTION
    SCRATCHPAD
    FINAL CALL
    DISALLOW PARALLEL
    NO COLLID
    ASUTIME LIMIT 10
    STAY RESIDENT NO
    PROGRAM TYPE SUB
    WLM ENVIRONMENT WLMENV
    SECURITY DB2
    NO DBINFO;

  GRANT EXECUTE ON SPECIFIC FUNCTION DSN8.DSN8DUADV,
                                     DSN8.DSN8DUCDVVV,
                                     DSN8.DSN8DUCDDVV,
                                     DSN8.DSN8DUATV,
                                     DSN8.DSN8DUCTVVV,
                                     DSN8.DSN8DUCTTVV,
                                     DSN8.DSN8DUCYFV,
                                     DSN8.DSN8DUCYFVV,
                                     DSN8.DSN8EUDNV,
                                     DSN8.DSN8EUDND,
                                     DSN8.DSN8EUMNV,
                                     DSN8.DSN8EUMND,
                                     DSN8.DSN8DUTINV,
                                     DSN8.DSN8DUTINVV,
                                     DSN8.DSN8DUTINVVV,
                                     DSN8.DSN8DUTISV,
                                     DSN8.DSN8DUTISVV,
                                     DSN8.DSN8DUTISVVV,
                                     DSN8.DSN8DUTILV,
                                     DSN8.DSN8DUTILVV,
                                     DSN8.DSN8DUTILVVV,
                                     DSN8.DSN8DUWFV
              TO PUBLIC;

//*
//*        STEP  3: PREPARE EXTERNAL FOR CURRENT DATE ALTDATE UDF
//*
//PH02US03 EXEC DSNHC,MEM=DSN8DUAD,COND=(4,LT),
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//             SOURCE,XREF),
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
```

```
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUAD),                    06100000
//             DISP=SHR                                                 06110000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               06120000
//             DISP=SHR                                                 06130000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUAD),                        06140000
//             DISP=SHR                                                 06150000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUAD),                     06160000
//             DISP=SHR                                                 06170000
//LKED.SYSIN   DD *                                                     06180000
 INCLUDE SYSLIB(DSNRLI)                                                 06190000
 NAME DSN8DUAD(R)                                                       06200000
//*                                                                     06210000
//*        STEP  4: PREPARE EXTERNAL FOR GIVEN DATE ALTDATE UDF         06220000
//*                                                                     06230000
//PH02US04 EXEC DSNHC,MEM=DSN8DUCD,COND=(4,LT),                         06240000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     06250000
//             SOURCE,XREF),                                            06260000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  06270000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 06280000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUCD),                    06290000
//             DISP=SHR                                                 06300000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               06310000
//             DISP=SHR                                                 06320000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUCD),                        06330000
//             DISP=SHR                                                 06340000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCD),                     06350000
//             DISP=SHR                                                 06360000
//LKED.SYSIN   DD *                                                     06370000
 INCLUDE SYSLIB(DSNRLI)                                                 06380000
 NAME DSN8DUCD(R)                                                       06390000
//*                                                                     06400000
//*        STEP  5: PREPARE EXTERNAL FOR CURRENT TIME ALTTIME UDF       06410000
//*                                                                     06420000
//PH02US05 EXEC DSNHC,MEM=DSN8DUAT,COND=(4,LT),                         06430000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     06440000
//             SOURCE,XREF),                                            06450000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  06460000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 06470000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUAT),                    06480000
//             DISP=SHR                                                 06490000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               06500000
//             DISP=SHR                                                 06510000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUAT),                        06520000
//             DISP=SHR                                                 06530000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUAT),                     06540000
//             DISP=SHR                                                 06550000
//LKED.SYSIN   DD *                                                     06560000
 INCLUDE SYSLIB(DSNRLI)                                                 06570000
 NAME DSN8DUAT(R)                                                       06580000
//*                                                                     06590000
//*        STEP  6: PREPARE EXTERNAL FOR GIVEN TIME ALTTIME UDF         06600000
//*                                                                     06610000
//PH02US06 EXEC DSNHC,MEM=DSN8DUCT,COND=(4,LT),                         06620000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     06630000
//             SOURCE,XREF),                                            06640000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  06650000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 06660000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUCT),                    06670000
//             DISP=SHR                                                 06680000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               06690000
//             DISP=SHR                                                 06700000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUCT),                        06710000
//             DISP=SHR                                                 06720000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCT),                     06730000
//             DISP=SHR                                                 06740000
//LKED.SYSIN   DD *                                                     06750000
 INCLUDE SYSLIB(DSNRLI)                                                 06760000
 NAME DSN8DUCT(R)                                                       06770000
//*                                                                     06780000
//*        STEP  7: PREPARE EXTERNAL FOR CURRENCY UDF                   06790000
//*                                                                     06800000
//PH02US07 EXEC DSNHC,MEM=DSN8DUCY,COND=(4,LT),                         06810000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     06820000
//             SOURCE,XREF),                                            06830000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  06840000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 06850000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUCY),                    06860000
//             DISP=SHR                                                 06870000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               06880000
//             DISP=SHR                                                 06890000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUCY),                        06900000
//             DISP=SHR                                                 06910000
```

```
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCY),                      06920000
//              DISP=SHR                                                06930000
//LKED.SYSIN   DD *                                                     06940000
 INCLUDE SYSLIB(DSNRLI)                                                 06950000
 NAME DSN8DUCY(R)                                                       06960000
//*                                                                     06970000
//*        STEP  8: PREPARE EXTERNAL FOR DAYNAME UDF                    06980000
//*                                                                     06990000
//PH02US08 EXEC DSNHCPP,MEM=DSN8EUDN,COND=(4,LT),                       07000000
//         PARM.PC=('HOST(CPP),CCSID(1047),MARGINS(1,80),STDSQL(NO)',   07010000
//              SOURCE,XREF),                                           07020000
//         PARM.CP=('/CXX SOURCE XREF OPTFILE(DD:CCOPTS)',              07030000
//              'LANGLVL(EXTENDED)'),                                   07040000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 07050000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EUDN),                    07060000
//              DISP=SHR                                                07070000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               07080000
//              DISP=SHR                                                07090000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8EUDN),                        07100000
//              DISP=SHR                                                07110000
//CP.CCOPTS    DD DSN=SYS1.PROCLIB(DSNHCPPS),DISP=SHR                   07120000
//CP.USERLIB   DD DSN=DSN!!0.SRCLIB.DATA,                               07130000
//              DISP=SHR                                                07140000
//PLKED.SYSDEFSD DD  DUMMY                                              07150000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EUDN),                      07160000
//              DISP=SHR                                                07170000
//LKED.SYSIN   DD *                                                     07180000
 INCLUDE SYSLIB(DSNRLI)                                                 07190000
 NAME DSN8EUDN(R)                                                       07200000
//*                                                                     07210000
//*        STEP  9: PREPARE EXTERNAL FOR MONTHNAME UDF                  07220000
//*                                                                     07230000
//PH02US09 EXEC DSNHCPP,MEM=DSN8EUMN,COND=(4,LT),                       07240000
//         PARM.PC=('HOST(CPP),CCSID(1047),MARGINS(1,80),STDSQL(NO)',   07250000
//              SOURCE,XREF),                                           07260000
//         PARM.CP=('/CXX SOURCE XREF OPTFILE(DD:CCOPTS)',              07270000
//              'LANGLVL(EXTENDED)'),                                   07280000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 07290000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8EUMN),                    07300000
//              DISP=SHR                                                07310000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               07320000
//              DISP=SHR                                                07330000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8EUMN),                        07340000
//              DISP=SHR                                                07350000
//CP.CCOPTS    DD DSN=SYS1.PROCLIB(DSNHCPPS),DISP=SHR                   07360000
//CP.USERLIB   DD DSN=DSN!!0.SRCLIB.DATA,                               07370000
//              DISP=SHR                                                07380000
//PLKED.SYSDEFSD DD  DUMMY                                              07390000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EUMN),                      07400000
//              DISP=SHR                                                07410000
//LKED.SYSIN   DD *                                                     07420000
 INCLUDE SYSLIB(DSNRLI)                                                 07430000
 NAME DSN8EUMN(R)                                                       07440000
//*                                                                     07450000
//*        STEP 10: PREPARE EXTERNAL FOR TABLE_NAME, TABLE_SCHEMA,      07460000
//*                  AND TABLE_LOCATION UDF'S                           07470000
//*                                                                     07480000
//PH02US10 EXEC DSNHC,MEM=DSN8DUTI,COND=(4,LT),                         07490000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     07500000
//              SOURCE,XREF),                                           07510000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  07520000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 07530000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUTI),                    07540000
//              DISP=SHR                                                07550000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               07560000
//              DISP=SHR                                                07570000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUTI),                        07580000
//              DISP=SHR                                                07590000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUTI),                      07600000
//              DISP=SHR                                                07610000
//LKED.IGNORE  DD *                                                     07620000
//LKED.SYSIN   DD *                                                     07630000
 INCLUDE SYSLIB(DSNRLI)                                                 07640000
 NAME DSN8DUTI(R)                                                       07650000
//*                                                                     07660000
//*        STEP 11: BIND PACKAGE FOR TABLE_NAME, TABLE_SCHEMA, AND      07670000
//*                  TABLE_LOCATION UDF'S                               07680000
//*                                                                     07690000
//PH02US11 EXEC PGM=IKJEFT01,COND=(4,LT)                                07700000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                         07710000
//SYSTSPRT DD  SYSOUT=*                                                 07720000
//SYSPRINT DD  SYSOUT=*                                                 07730000
```

```
//CEEDUMP  DD  SYSOUT=*                                                 07740000
//SYSUDUMP DD  SYSOUT=*                                                 07750000
//SYSOUT   DD  SYSOUT=*                                                 07760000
//REPORT   DD  SYSOUT=*                                                 07770000
//SYSIN    DD  *                                                        07780000
//SYSTSIN  DD  *                                                        07790000
 DSN SYSTEM(DSN)                                                        07800000
 BIND PACKAGE (DSN8DU!!) MEMBER(DSN8DUTI) -                             07810000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                07820000
 END                                                                    07830000
//*                                                                     07840000
//*        STEP 12: EXERCISE THE SAMPLE UDF'S                           07850000
//*                                                                     07860000
//PH02US12 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20                    07870000
//SYSTSPRT DD  SYSOUT=*                                                 07880000
//SYSPRINT DD  SYSOUT=*                                                 07890000
//SYSUDUMP DD  SYSOUT=*                                                 07900000
//SYSTSIN  DD  *                                                        07910000
 DSN SYSTEM(DSN)                                                        07920000
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP!!) -                                 07930000
      LIB('DSN!!0.RUNLIB.LOAD') PARMS('/ALIGN(MID)')                    07940000
 END                                                                    07950000
//SYSIN    DD  *                                                        07950104
 SET CURRENT SQLID = 'SYSADM';                                          07951004
//         DD  DSN=DSN!!0.SDSNSAMP(DSNTESU),                            07960004
//             DISP=SHR                                                 07970000
//*                                                                     07980000
//*        STEP 13: PREPARE EXTERNAL FOR WEATHER UDF TABLE FUNCTION     07990000
//*                                                                     08000000
//PH02US13 EXEC DSNHC,MEM=DSN8DUWF,COND=(4,LT),                         08010000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     08020000
//             SOURCE,XREF),                                            08030000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  08040000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 08050000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUWF),                    08060000
//             DISP=SHR                                                 08070000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               08080000
//             DISP=SHR                                                 08090000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUWF),                        08100000
//             DISP=SHR                                                 08110000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUWF),                     08120000
//             DISP=SHR                                                 08130000
//LKED.IGNORE  DD *                                                     08140000
//LKED.SYSIN   DD *                                                     08150000
 INCLUDE SYSLIB(DSNRLI)                                                 08160000
 NAME DSN8DUWF(R)                                                       08170000
//*                                                                     08180000
//*        STEP 14: PREPARE CLIENT FOR WEATHER UDF TABLE FUNCTION       08190000
//*                                                                     08200000
//PH02US14 EXEC DSNHC,MEM=DSN8DUWC,COND=(4,LT),                         08210000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     08220000
//             SOURCE,XREF),                                            08230000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  08240000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 08250000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUWC),                    08260000
//             DISP=SHR                                                 08270000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               08280000
//             DISP=SHR                                                 08290000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DUWC),                        08300000
//             DISP=SHR                                                 08310000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUWC),                     08320000
//             DISP=SHR                                                 08330000
//LKED.IGNORE  DD *                                                     08340000
//LKED.SYSIN   DD *                                                     08350000
 INCLUDE SYSLIB(DSNELI)                                                 08360000
 INCLUDE SYSLIB(DSNTIAR)                                                08370000
 NAME DSN8DUWC(R)                                                       08380000
//*                                                                     08390000
//*        STEP 15: BIND PACKAGE & PLAN FOR WEATHER TBL FUNC CLIENT     08400000
//*                                                                     08410000
//PH02US15 EXEC PGM=IKJEFT01,COND=(4,LT)                                08420000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                         08430000
//SYSTSPRT DD  SYSOUT=*                                                 08440000
//SYSPRINT DD  SYSOUT=*                                                 08450000
//CEEDUMP  DD  SYSOUT=*                                                 08460000
//SYSUDUMP DD  SYSOUT=*                                                 08470000
//SYSOUT   DD  SYSOUT=*                                                 08480000
//REPORT   DD  SYSOUT=*                                                 08490000
//SYSTSIN  DD  *                                                        08510000
 DSN SYSTEM(DSN)                                                        08520000
 BIND PACKAGE (DSN8DU!!) MEMBER(DSN8DUWC) -                             08530000
      ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                08540000
```

```
 BIND PLAN (DSN8UW!!) PKLIST(DSN8DU!!.*) -                              08550000
      ACTION(REPLACE) RETAIN +                                          08551000
      ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)           08560000
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  08570000
     LIB('DSN!!0.RUNLIB.LOAD')                                          08580000
 END                                                                    08590000
//SYSIN    DD  *                                                        08600000
  SET CURRENT SQLID = 'SYSADM';                                         08601001
  GRANT EXECUTE,BIND ON PLAN DSN8UW!!                                   08610003
    TO PUBLIC;                                                          08611005
//*                                                                     08620000
//*        STEP 16: INVOKE THE SAMPLE UDF TABLE CLIENT                  08630000
//*                                                                     08640000
//PH02US16 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20                    08650000
//SYSTSPRT DD  SYSOUT=*                                                 08660000
//SYSPRINT DD  SYSOUT=*                                                 08670000
//SYSUDUMP DD  SYSOUT=*                                                 08680000
//SYSTSIN  DD  *                                                        08690000
 DSN SYSTEM(DSN)                                                        08700000
 RUN  PROGRAM(DSN8DUWC) PLAN(DSN8UW!!) -                                08710000
      LIB('DSN!!0.RUNLIB.LOAD') -                                       08720000
      PARMS('DSN!!0.SDSNIVPD(DSN8LWC')                                  08730000
 END                                                                    08740000
//*                                                                     08750000
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ71

PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT OF THE Db2 LOB SAMPLE C APPLICATION.

```
//******************************************************************** 00010000
//*  NAME = DSNTEJ71                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 7                                         00050000
//*                     SAMPLE APPLICATIONS: POPULATE, CHECK LOB TABLE  00060000
//*                     C LANGUAGE                                      00070000
//*                                                                     00080000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00090000
//*    5605-DB2                                                         00100000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.         00110000
//*                                                                     00120000
//*    STATUS = VERSION 11                                              00130000
//*                                                                     00140000
//*  FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT     00150000
//*             OF THE DB2 LOB SAMPLE C APPLICATION:                    00160000
//*             - DSN8DLPL: POPULATES THE PSEG AND BMP IMAGE COLUMNS    00170000
//*                         IN THE DSN8!!0.EMP_PHOTO_RESUME SAMPLE LOB  00180000
//*                         TABLE.  THE INPUT DATA IS READ FROM A TSO   00190000
//*                         DATA SET.  THIS PROGRAM DEMONSTRATES HOW    00200000
//*                         TO POPULATE LOB COLUMNS WITH MORE THAN 32   00210000
//*                         KB OF DATA.                                 00220000
//*                                                                     00230000
//*             - DSN8DLTC: VALIDATES THE CONTENTS OF THE LOB COLUMNS   00240000
//*                         IN THE DSN8!!0.EMP_PHOTO_RESUME TABLE.      00250000
//*                         THIS IS DONE BY COMPARING THE DATA IN THE   00260000
//*                         TABLE TO THE SOURCE DATA SETS.              00270000
//*                                                                     00280000
//*  CHANGE ACTIVITY =                                                  00290000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00291000
//*               ADD SET CURRENT SQLID                                 00291101
//*                                                                     00292000
//******************************************************************** 00300000
//JOBLIB  DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR                              00310000
//        DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00320000
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                           00330000
//        DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                           00340000
//*                                                                     00350000
//*        STEP  1: PREPARE LOADER FOR EMPLOYEE PHOTO IMAGES            00360000
//*                                                                     00370000
//PH071S01 EXEC DSNHC,MEM=DSN8DLPL,COND=(4,LT),                         00380000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00390000
//             SOURCE,XREF),                                            00400000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  00410000
```

```
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 00420000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DLPL),                    00430000
//             DISP=SHR                                                 00440000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00450000
//             DISP=SHR                                                 00460000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DLPL),                        00470000
//             DISP=SHR                                                 00480000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLPL),                     00490000
//             DISP=SHR                                                 00500000
//LKED.SYSIN   DD *                                                     00510000
 INCLUDE SYSLIB(DSNELI)                                                 00520000
 NAME DSN8DLPL(R)                                                       00530000
//*                                                                     00540000
//*        STEP  2: PREPARE SAMPLE LOB TABLE VALIDATOR                  00550000
//*                                                                     00560000
//PH071S02 EXEC DSNHC,MEM=DSN8DLTC,COND=(4,LT),                         00570000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',     00580000
//             SOURCE,XREF),                                            00590000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  00600000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                 00610000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DLTC),                    00620000
//             DISP=SHR                                                 00630000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00640000
//             DISP=SHR                                                 00650000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DLTC),                        00660000
//             DISP=SHR                                                 00670000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLTC),                     00680000
//             DISP=SHR                                                 00690000
//LKED.SYSIN   DD *                                                     00700000
 INCLUDE SYSLIB(DSNELI)                                                 00710000
 NAME DSN8DLTC(R)                                                       00720000
//*                                                                     00730000
//*        STEP  3: BIND PACKAGES AND PLANS FOR DSN8DLPL AND DSN8DLTC   00740000
//*                                                                     00750000
//PH071S03 EXEC PGM=IKJEFT01,COND=(4,LT)                                00760000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                         00770000
//SYSTSPRT DD  SYSOUT=*                                                 00780000
//SYSPRINT DD  SYSOUT=*                                                 00790000
//CEEDUMP  DD  SYSOUT=*                                                 00800000
//SYSUDUMP DD  SYSOUT=*                                                 00810000
//SYSOUT   DD  SYSOUT=*                                                 00820000
//REPORT   DD  SYSOUT=*                                                 00830000
//SYSIN    DD  *                                                        00840000
 SET CURRENT SQLID = 'SYSADM';                                          00841001
 GRANT BIND, EXECUTE ON PLAN DSN8LC!!, DSN8LL!!                         00850002
  TO PUBLIC;                                                            00851002
//SYSTSIN  DD  *                                                        00860000
 DSN SYSTEM(DSN)                                                        00870000
  BIND PACKAGE (DSN8LL!!) MEMBER(DSN8DLPL) -                            00880000
       QUALIFIER(DSN8!!0) -                                             00890000
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)               00900000
  BIND PLAN(DSN8LL!!) PKLIST(DSN8LL!!.*) -                              00910000
       ACTION(REPLACE) RETAIN +                                         00911000
       ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)          00920000
                                                                        00930000
  BIND PACKAGE (DSN8LC!!) MEMBER(DSN8DLTC) -                            00940000
       QUALIFIER(DSN8!!0) -                                             00950000
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)               00960000
  BIND PLAN(DSN8LC!!) PKLIST(DSN8LC!!.*) -                              00970000
       ACTION(REPLACE) RETAIN +                                         00971000
       ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)          00980000
                                                                        00990000
  RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                01000000
       LIB('DSN!!0.RUNLIB.LOAD')                                        01010000
 END                                                                    01020000
//*                                                                     01030000
//*        STEP  4: LOAD SAMPLE EMPLOYEE PHOTO IMAGES                   01040000
//*                                                                     01050000
//PH071S04 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20                    01060000
//SYSTSPRT DD  SYSOUT=*                                                 01070000
//SYSPRINT DD  SYSOUT=*                                                 01080000
//CEEDUMP  DD  SYSOUT=*                                                 01090000
//SYSABEND DD  SYSOUT=*                                                 01100000
//SYSUDUMP DD  SYSOUT=*                                                 01110000
//SYSOUT   DD  SYSOUT=*                                                 01120000
//SYSTSIN  DD  *                                                        01130000
 DSN SYSTEM(DSN)                                                        01140000
 RUN PROGRAM(DSN8DLPL) PLAN(DSN8LL!!) -                                 01150000
     LIB('DSN!!0.RUNLIB.LOAD')                                          01160000
 END                                                                    01170000
//PSEGIN00 DD  DSN=DSN!!0.SDSNIVPD(DSN8P130),DISP=SHR                   01180000
//BMPIN00  DD  DSN=DSN!!0.SDSNIVPD(DSN8B130),DISP=SHR                   01190000
```

```
//PSEGIN01 DD  DSN=DSN!!0.SDSNIVPD(DSN8P140),DISP=SHR
//BMPIN01  DD  DSN=DSN!!0.SDSNIVPD(DSN8B140),DISP=SHR
//PSEGIN02 DD  DSN=DSN!!0.SDSNIVPD(DSN8P150),DISP=SHR
//BMPIN02  DD  DSN=DSN!!0.SDSNIVPD(DSN8B150),DISP=SHR
//PSEGIN03 DD  DSN=DSN!!0.SDSNIVPD(DSN8P190),DISP=SHR
//BMPIN03  DD  DSN=DSN!!0.SDSNIVPD(DSN8B190),DISP=SHR
//*
//*        STEP  5: VERIFY THE CONTENTS OF THE SAMPLE LOB TABLE
//*
//PH071S05 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8DLTC) PLAN(DSN8LC!!)
 END
//PSEGIN00 DD  DSN=DSN!!0.SDSNIVPD(DSN8P130),DISP=SHR
//BMPIN00  DD  DSN=DSN!!0.SDSNIVPD(DSN8B130),DISP=SHR
//RESUME00 DD  DSN=DSN!!0.SDSNIVPD(DSN8R130),DISP=SHR
//PSEGIN01 DD  DSN=DSN!!0.SDSNIVPD(DSN8P140),DISP=SHR
//BMPIN01  DD  DSN=DSN!!0.SDSNIVPD(DSN8B140),DISP=SHR
//RESUME01 DD  DSN=DSN!!0.SDSNIVPD(DSN8R140),DISP=SHR
//PSEGIN02 DD  DSN=DSN!!0.SDSNIVPD(DSN8P150),DISP=SHR
//BMPIN02  DD  DSN=DSN!!0.SDSNIVPD(DSN8B150),DISP=SHR
//RESUME02 DD  DSN=DSN!!0.SDSNIVPD(DSN8R150),DISP=SHR
//PSEGIN03 DD  DSN=DSN!!0.SDSNIVPD(DSN8P190),DISP=SHR
//BMPIN03  DD  DSN=DSN!!0.SDSNIVPD(DSN8B190),DISP=SHR
//RESUME03 DD  DSN=DSN!!0.SDSNIVPD(DSN8R190),DISP=SHR
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

DSNTEJ73

PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT OF THE Db2 LOB SAMPLE C APPLICATION.

```
//*********************************************************************
//*  NAME = DSNTEJ73
//*
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//*                     PHASE 7
//*                     SAMPLE APPLICATIONS: VIEW, MANIPULATE CLOB DATA
//*                     C LANGUAGE
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 11
//*
//*  FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT
//*             OF THE DB2 LOB SAMPLE C APPLICATION:
//*             -  DSN8SDM: CAF CONNECTION MANAGER (C LANGUAGE), USED
//*                         TO INVOKE THE DB2 SAMPLE APPLICATIONS MENU
//*                         UNDER ISPF AND TO MANAGE INVOKATION OF THE
//*                         DB2 SAMPLE APPLICATION PROGRAMS, INCLUDING
//*                         THE LOB SAMPLE RESUME AND PHOTO IMAGE
//*                         VIEWERS.
//*
//*             - DSN8DLRV: EXTRACTS A SPECIFIED EMPLOYEE'S RESUME IN
//*                         CLOB FORMAT FROM DSN8!!0.EMP_PHOTO_RESUME.
//*                         DB2 LOB LOCATOR FUNCTIONS ARE USED TO PARSE
//*                         DATA FROM CLOB FORMAT INTO ISPF FIELDS AND
//*                         THE RESULT IS DISPLAYED TO THE USER.
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651
//*               ADD SET CURRENT SQLID
//*
//*********************************************************************
//JOBLIB  DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR
//        DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR
```

```
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                          00340000
//        DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                          00350000
//*                                                                    00360000
//*        STEP  1: PREPARE SAMPLE CALL ATTACH CONTROLLER              00370000
//*                                                                    00380000
//PH073S01 EXEC DSNHC,MEM=DSN8SDM,COND=(4,LT),                         00390000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',    00400000
//             SOURCE,XREF),                                           00410000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)', 00420000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                00430000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SDM),                    00440000
//             DISP=SHR                                                00450000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                              00460000
//             DISP=SHR                                                00470000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8SDM),                        00480000
//             DISP=SHR                                                00490000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SDM),                     00500000
//             DISP=SHR                                                00510000
//LKED.SYSIN   DD *                                                    00520000
 INCLUDE SYSLIB(DSNALI)                                                00530000
 NAME DSN8SDM(R)                                                       00540000
//*                                                                    00550000
//*        STEP  2: PREPARE EMPLOYEE RESUME VIEWER (ISPF)              00560000
//*                                                                    00570000
//PH073S02 EXEC DSNHC,MEM=DSN8DLRV,COND=(4,LT),                        00580000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',    00590000
//             SOURCE,XREF),                                           00600000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)', 00610000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                00620000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DLRV),                   00630000
//             DISP=SHR                                                00640000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                              00650000
//             DISP=SHR                                                00660000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DLRV),                       00670000
//             DISP=SHR                                                00680000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLRV),                    00690000
//             DISP=SHR                                                00700000
//LKED.SYSIN   DD *                                                    00710000
 INCLUDE SYSLIB(DSNALI)                                                00720000
 NAME DSN8DLRV(R)                                                      00730000
//*                                                                    00740000
//*        STEP  3: BIND PACKAGE AND PLAN FOR THE RESUME VIEWER        00750000
//*                                                                    00760000
//PH073S03 EXEC PGM=IKJEFT01,COND=(4,LT)                               00770000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                        00780000
//SYSTSPRT DD  SYSOUT=*                                                00790000
//SYSPRINT DD  SYSOUT=*                                                00800000
//CEEDUMP  DD  SYSOUT=*                                                00810000
//SYSUDUMP DD  SYSOUT=*                                                00820000
//SYSOUT   DD  SYSOUT=*                                                00830000
//REPORT   DD  SYSOUT=*                                                00840000
//SYSTSIN  DD  *                                                       00850000
 DSN SYSTEM(DSN)                                                       00860000
  BIND PACKAGE (DSN8LR!!) -                                            00870000
       MEMBER(DSN8DLRV) -                                              00880000
       QUALIFIER(DSN8!!0) -                                            00890000
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)              00900000
  BIND PLAN(DSN8LR!!) -                                                00910000
       PKLIST(DSN8LR!!.*) -                                            00920000
       ACTION(REPLACE) RETAIN +                                        00921000
       ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)         00930000
  RUN  PROGRAM(DSNTIAD) -                                              00940000
       PLAN(DSNTIA!!)  -                                               00950000
       LIB('DSN!!0.RUNLIB.LOAD')                                       00960000
 END                                                                   00970000
//SYSIN    DD  *                                                       00980000
  SET CURRENT SQLID = 'SYSADM';                                        00981001
  GRANT EXECUTE,BIND ON PLAN DSN8LR!!                                  00990002
    TO PUBLIC;                                                         01000003
```

Related reference

"Sample applications in TSO" on page 1013

A set of Db2 sample applications run in the TSO environment.

DSNTEJ75

PREPARES AND RUNS THE FOLLOWING PROGRAM IN SUPPORT OF THE Db2 LOB SAMPLE C APPLICATION.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ75                                                     00020000
//*                                                                      00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                           00040000
//*                     PHASE 7                                          00050000
//*                     SAMPLE APPLICATIONS: VIEW, MANIPULATE BLOB DATA 00060000
//*                     C LANGUAGE                                       00070000
//*                                                                      00080000
//*    LICENSED MATERIALS - PROPERTY OF IBM                              00090000
//*    5605-DB2                                                          00100000
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.          00110000
//*                                                                      00120000
//*    STATUS = VERSION 11                                               00130000
//*                                                                      00140000
//*  FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAM IN SUPPORT       00150000
//*             OF THE DB2 LOB SAMPLE C APPLICATION:                     00160000
//*             - DSN8DLPV: EXTRACTS A SPECIFIED EMPLOYEE'S PSEG PHOTO  00170000
//*                         IMAGE IN BLOB FORMAT FROM THE SMAPLE TABLE  00180000
//*                         DSN8!!0.EMP_PHOTO_RESUME.  THE DATA IS       00190000
//*                         HANDED OFF TO GDDM FOR CONVERSION FOR CON-  00200000
//*                         VERSION AND DISPLAY.                         00210000
//*                                                                      00220000
//*  CHANGE ACTIVITY =                                                   00230000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00231000
//*               ADD SET CURRENT SQLID                                  00231101
//*                                                                      00232000
//********************************************************************* 00240000
//JOBLIB  DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR                               00250000
//        DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                               00260000
//        DD  DSN=CEE.V!R!M!.SCEERUN,DISP=SHR                            00270000
//        DD  DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                            00280000
//*                                                                      00290000
//*        STEP  1: PREPARE EMPLOYEE PHOTO VIEWER (GDDM)                 00300000
//*                                                                      00310000
//PH075S01 EXEC DSNHC,MEM=DSN8DLPV,COND=(4,LT),                          00320000
//         PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',      00330000
//             SOURCE,XREF),                                             00340000
//         PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',  00350000
//         PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'                  00360000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DLPV),                     00370000
//             DISP=SHR                                                  00380000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                                00390000
//             DISP=SHR                                                  00400000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8DLPV),                         00410000
//             DISP=SHR                                                  00420000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLPV),                      00430000
//             DISP=SHR                                                  00440000
//LKED.SYSIN   DD *                                                      00450000
 INCLUDE SYSLIB(ADMASRT)                                                 00460000
 INCLUDE SYSLIB(DSNTIAR)                                                 00470000
 INCLUDE SYSLIB(DSNALI)                                                  00480000
 NAME DSN8DLPV(R)                                                        00490000
//*                                                                      00500000
//*        STEP  2: BIND PACKAGE AND PLAN FOR THE PHOTO VIEWER           00510000
//*                                                                      00520000
//PH075S02 EXEC PGM=IKJEFT01,COND=(4,LT)                                 00530000
//DBRMLIB  DD  DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                          00540000
//SYSTSPRT DD  SYSOUT=*                                                  00550000
//SYSPRINT DD  SYSOUT=*                                                  00560000
//CEEDUMP  DD  SYSOUT=*                                                  00570000
//SYSUDUMP DD  SYSOUT=*                                                  00580000
//SYSOUT   DD  SYSOUT=*                                                  00590000
//REPORT   DD  SYSOUT=*                                                  00600000
//SYSTSIN  DD  *                                                         00610000
 DSN SYSTEM(DSN)                                                         00620000
  BIND PACKAGE (DSN8LP!!) -                                              00630000
       MEMBER(DSN8DLPV) -                                                00640000
       QUALIFIER(DSN8!!0) -                                              00650000
       ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                00660000
  BIND PLAN(DSN8LP!!) -                                                  00670000
       PKLIST(DSN8LP!!.*) -                                              00680000
       ACTION(REPLACE) RETAIN +                                          00681000
```

```
       ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)            00690000
  RUN  PROGRAM(DSNTIAD) -                                                 00700000
       PLAN(DSNTIA!!) -                                                   00710000
       LIB('DSN!!0.RUNLIB.LOAD')                                          00720000
 END                                                                      00730000
//SYSIN  DD  *                                                            00740000
 SET CURRENT SQLID = 'SYSADM';                                            00741001
 GRANT EXECUTE ON PLAN DSN8LP!!                                           00750002
   TO PUBLIC;                                                             00760002
```

Related reference

"Sample applications in TSO" on page 1013
A set of Db2 sample applications run in the TSO environment.

Sample applications in IMS

A set of Db2 sample applications run in the IMS environment.

Table 183. Sample Db2 applications for IMS

Application	Program name	JCL member name	Description
Organization	DSN8IC0 DSN8IC1 DSN8IC2	DSNTEJ4C	IMS COBOL Organization Application
Organization	DSN8IP0 DSN8IP1 DSN8IP2	DSNTEJ4P	IMS PL/I Organization Application
Project	DSN8IP6 DSN8IP7 DSN8IP8	DSNTEJ4P	IMS PL/I Project Application
Phone	DSN8IP3	DSNTEJ4P	IMS PL/I Phone Application. This program lists employee telephone numbers and updates them if requested.

Related reference

Data sets that the precompiler uses
When you invoke the precompiler you need to provide data sets that contain input for the precompiler, such as the host programming statements and SQL statements. You also need to provide data sets where the precompiler can store its output, such as the modified source code and diagnostics messages.

DSN8IC0

THIS MODULE RECEIVES AN INPUT MESSAGE AND DEFORMATS IT, CALLS DSN8IC1, FORMATS OUTPUT MESSAGE AND SENDS IT.

```
        IDENTIFICATION DIVISION.                                          00010000
       *------------------------                                          00012000
        PROGRAM-ID. DSN8IC0.                                              00014000
                                                                          00016000
       ******* DSN8IC0 - IMS SUBSYSTEM INTERFACE MODULE - COBOL ******** 00018000
       *                                                               * 00020000
       *   MODULE NAME = DSN8IC0                                       * 00030000
       *                                                               * 00040000
       *   DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                   * 00050000
       *                      SUBSYSTEM INTERFACE MODULE               * 00060000
       *                      IMS                                      * 00070000
       *                      COBOL                                    * 00080000
```

```
*                      ORGANIZATION APPLICATION                * 00090000
*                                                              * 00100000
*LICENSED MATERIALS - PROPERTY OF IBM                          * 00110000
*5615-DB2                                                      * 00116000
*(C) COPYRIGHT 1996, 2013 IBM CORP.  ALL RIGHTS RESERVED.      * 00125000
*                                                              * 00126000
*STATUS = VERSION 11                                           * 00128001
*                                                              * 00130000
*   FUNCTION = THIS MODULE RECEIVES AN INPUT MESSAGE AND       * 00160000
*              DEFORMATS IT, CALLS DSN8IC1,                    * 00170000
*              FORMATS OUTPUT MESSAGE AND SENDS IT.            * 00180000
*                                                              * 00190000
*   NOTES = NONE                                               * 00200000
*                                                              * 00210000
*   MODULE TYPE =                                              * 00220000
*      PROCESSOR   = DB2 PREPROCESSOR, COBOL COMPILER          * 00230000
*      MODULE SIZE = SEE LINKEDIT                              * 00240000
*      ATTRIBUTES  = REUSABLE                                  * 00250000
*                                                              * 00260000
*   ENTRY POINT =  DSN8IC0                                     * 00270000
*      PURPOSE = SEE FUNCTION                                  * 00280000
*      LINKAGE = FROM IMS                                      * 00290000
*                                                              * 00300000
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:  * 00310000
*                                                              * 00320000
*              SYMBOLIC LABEL/NAME  =  DSN8ICGI                * 00330000
*              DESCRIPTION = IMS/VS MFS GENERAL MENU           * 00340000
*                                                              * 00350000
*              SYMBOLIC LABEL/NAME  =  DSN8ICDI                * 00360000
*              DESCRIPTION = IMS/VS MFS DETAIL MENU            * 00370000
*                                                              * 00380000
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                * 00390000
*                                                              * 00400000
*              SYMBOLIC LABEL/NAME  =  DSN8ICGO                * 00410000
*              DESCRIPTION = IMS/VS MFS GENERAL MENU           * 00420000
*                                                              * 00430000
*              SYMBOLIC LABEL/NAME  =  DSN8ICDO                * 00440000
*              DESCRIPTION = IMS/VS MFS DETAIL MENU            * 00450000
*                                                              * 00460000
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION              * 00470000
*                                                              * 00480000
*   EXIT-ERROR =                                               * 00490000
*                                                              * 00500000
*      RETURN CODE = NONE                                      * 00510000
*                                                              * 00520000
*      ABEND CODES =  NONE                                     * 00530000
*                                                              * 00540000
*      ERROR-MESSAGES =                                        * 00550000
*        DSN8064E - INVALID DL/I STC-CODE ON GU MSG            * 00560000
*        DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG          * 00570000
*                                                              * 00580000
*   EXTERNAL REFERENCES =                                      * 00590000
*      ROUTINES/SERVICES =  MODULE DSN8IC1                     * 00600000
*                           MODULE CBLTDLI                     * 00610000
*                           MODULE DSN8MCG                     * 00620000
*      DATA-AREAS =                                            * 00630000
*         DSN8MCCA           - PARAMETER TO BE PASSED TO DSN8IC1* 00640000
*                              CONTAINS TERMINAL INPUT AND     * 00650000
*                              OUTPUT AREAS.                   * 00660000
*                                                              * 00670000
*      CONTROL-BLOCKS =                                        * 00680000
*         IN-MESSAGE         - MFS INPUT                       * 00690000
*         OUT-MESSAGE        - MFS OUTPUT                      * 00700000
*                                                              * 00710000
*   TABLES = NONE                                              * 00720000
*                                                              * 00730000
*   CHANGE-ACTIVITY =                                          * 00740000
*     05/18/2012: SWITCH ARITHMETICS FROM COMP TO COMP-5 PM66408* 00750002
*                                                              * 00751000
*                                                              * 00760000
*  *PSEUDOCODE*                                                * 00770000
*                                                              * 00780000
*    PROCEDURE                                                 * 00790000
*      DECLARATIONS.                                           * 00800000
*        ALLOCATE COBOL WORK AREA FOR COMMAREA.                * 00810000
*        INITIALIZATION.                                       * 00820000
*             PUT MODNAME 'DSN8ICGO' IN MODNAME FIELD.         * 00830000
*             PUT MODULE NAME 'DSN8IC0' IN AREA USED BY        * 00840000
*             ERROR-HANDLER.                                   * 00850000
*                                                              * 00860000
*      STEP1.                                                  * 00870000
*      CALL DLI GU INPUT MESSAGE.                              * 00880000
```

```
      *      IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND        * 00890000
      *         STOP PROGRAM.                                         * 00900000
      *                                                               * 00910000
      *      IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.      * 00920000
      *         MOVE INPUT MESSAGE FIELDS TO CORRESPONDING            * 00930000
      *              INAREA FIELDS IN COMPARM.                        * 00940000
      *         CALL DSN8IC1 (COMMAREA)                               * 00950000
      *         MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING      * 00960000
      *              OUTPUT MESSAGE FIELDS.                           * 00970000
      *         IF LASTSCR 'DSN8001' MOVE 'DSN8ICGO' TO MODNAME FIELD * 00980000
      *              ELSE MOVE 'DSN8ICDO' TO MODNAME FIELD.           * 00990000
      *                                                               * 01000000
      *      CALL DLI ISRT OUTPUT MESSAGE.                            * 01010000
      *      IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND        * 01020000
      *         STOP PROGRAM.                                         * 01030000
      *                                                               * 01040000
      *      END.                                                     * 01050000
      *                                                               * 01060000
      *************************************************************** 01070000
      *                                                                 01080000
       ENVIRONMENT DIVISION.                                            01130000
      *-----------------------                                          01140000
                                                                        01150000
       DATA DIVISION.                                                   01160000
      *-----------------------                                          01170000
       WORKING-STORAGE SECTION.                                         01180000
      ***************************************************************   01190000
      *    DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE     *  01200000
      *    SUBSYSTEM DEPENDENT MODULE IMS/DLI AND SQL1 AND SQL2      *  01210000
      ***************************************************************   01220000
      *                                                                 01230000
       01   COMMAREA.                                                   01240000
           EXEC SQL INCLUDE DSN8MCCA END-EXEC.                          01250000
      ***************************************************************   01260000
      *       DECLARATION FOR INPUT:  MIDNAME DSN8ICGI/DSN8ICDI      *  01270000
      ***************************************************************   01280000
      *                                                                 01290000
       01   IN-MESSAGE.                                                 01300000
            02 LL               PIC S9(3) COMP-5.                       01310000
            02 Z1               PIC X.                                  01320000
            02 Z2               PIC X.                                  01330000
            02 TC-CODE          PIC X(7).                               01340000
            02 IN-PUT.                                                  01350000
               03 MAJSYS     PIC X.                                     01360000
               03 ACTION     PIC X.                                     01370000
               03 OBJFLD     PIC X(2).                                  01380000
               03 SRCH       PIC X(2).                                  01390000
               03 PFKIN      PIC X(2).                                  01400000
               03 DATAIN     PIC X(60).                                 01410000
               03 TRANDATA   PIC X(40) OCCURS 15.                       01420000
      *                                                                 01430000
            02 IN-PUT0   REDEFINES IN-PUT  PIC X(668).                  01440000
      ***************************************************************   01450000
      *       DECLARATION FOR OUTPUT: MODNAME DSN8ICGO/DSN8ICDO      *  01460000
      ***************************************************************   01470000
      *                                                                 01480000
       01   OUT-MESSAGE.                                                01490000
            02 LL               PIC S9(3) COMP-5.                       01500000
            02 ZZ               PIC S9(3) COMP-5 VALUE +0.              01510000
            02 OUTPUTAREA.                                              01520000
               03 MAJSYS     PIC X.                                     01530000
               03 ACTION     PIC X.                                     01540000
               03 OBJFLD     PIC X(2).                                  01550000
               03 SRCH       PIC X(2).                                  01560000
               03 DATAOUT    PIC X(60).                                 01570000
               03 HTITLE     PIC X(50).                                 01580000
               03 DESC2      PIC X(50).                                 01590000
               03 DESC3      PIC X(50).                                 01600000
               03 DESC4      PIC X(50).                                 01610000
               03 MSG.                                                  01620000
                  05 STC     PIC X(4).                                  01630000
                  05 MSGTEXT PIC X(75).                                 01640000
               03 PFKTEXT    PIC X(79).                                 01650000
               03 OUTPUT0    OCCURS 15.                                 01660000
                  05 LINE0   PIC X(79).                                 01670000
      *                                                                 01680000
            02 OUTPUTAREA0 REDEFINES OUTPUTAREA PIC X(1609).            01690000
      ***************************************************************   01700000
      * FIELDS SENT TO MESSAGE ROUTINE                               *  01710000
      ***************************************************************   01720000
       01  MSGCODE              PIC X(04).                              01730000
                                                                        01740000
```

```
 01  OUTMSG               PIC X(69).
****************************************************************
*                DECLARATION FOR PGM-LOGIC                     *
****************************************************************
*
 77  GU-FKT               PIC X(4) VALUE 'GU  '.
 77  ISRT-FKT             PIC X(4) VALUE 'ISRT'.
 77  CHNG-FKT             PIC X(4) VALUE 'CHNG'.
 77  ROLL-FKT             PIC X(4) VALUE 'ROLL'.
*
 77  MODNAME              PIC X(8).
****************************************************************
*               LINKAGE SECTION                                *
****************************************************************
 LINKAGE SECTION.
****************************************************************
*           DECLARATION FOR IO / ALTPCB                        *
****************************************************************
*
 01  IOPCB.
     02 IOLTERM           PIC X(8).
     02 FILLER            PIC X(2).
     02 STC-CODE          PIC X(2).
     02 CDATE             PIC X(4).
     02 CTIME             PIC X(4).
     02 SEQNUM            PIC X(4).
     02 MOD-NAME          PIC X(8).
     02 USERID            PIC X(8).
*
 01  ALTPCB.
     02 ALTLTERM          PIC X(8).
     02 FILLER            PIC X(2).
     02 STC-CODE          PIC X(2).

 PROCEDURE DIVISION.
*---------------------
*
     ENTRY 'DLITCBL' USING IOPCB ALTPCB.
****************************************************************
*         ALLOCATE COBOL WORK AREA /INITIALIZATIONS            *
****************************************************************
*
 CSTART.
     MOVE SPACES     TO COMMAREA.
     MOVE SPACES     TO IN-MESSAGE.
     MOVE 'DSN8ICGO' TO MODNAME.
     MOVE 'DSN8IC0'  TO MAJOR IN DSN8-MODULE-NAME.
     MOVE 'O'        TO MAJSYS IN OUTAREA.
     MOVE '0'        TO EXITCODE.
     MOVE +1613      TO LL IN OUT-MESSAGE.
*
****************************************************************
*     CALL DL1 GU INPUT MESSAGE                                *
*     PRINT ERROR MESSAGE IF STATUS CODE NOT OK                *
****************************************************************

*                                         **CALL DL1 GU
     CALL 'CBLTDLI'
     USING GU-FKT IOPCB IN-MESSAGE.

*                                         **ERROR?
     IF STC-CODE IN IOPCB NOT = ' '
        THEN MOVE '064E' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO MSGTEXT IN OUTPUTAREA
             MOVE STC-CODE IN IOPCB TO STC  IN OUTPUTAREA
             GO TO CSEND.

****************************************************************
*     CLEARED AND UNFORMATTED SCREEN?                          *
****************************************************************

     IF Z2 = LOW-VALUE
        THEN MOVE '00' TO PFKIN   IN INAREA.
        MOVE IOLTERM IN IOPCB TO TRMID  IN CONVID.
        MOVE USERID  IN IOPCB TO USERID IN CONVID.

*                                       **MOVE INPUT MESSAGE
*                                       **FIELDS TO INAREA FIELDS
        MOVE IN-PUT0 TO INAREA0.
        MOVE 'O'        TO MAJSYS IN INAREA.
*
```

```
           CALL 'DSN8IC1' USING COMMAREA.

      *                                     **MOVE OUTAREA FIELDS TO
      *                                     **OUTPUT MESSAGE FIELDS
              MOVE OUTAREA0 TO OUTPUTAREA0.
      *
           IF LASTSCR = 'DSN8002'
              THEN MOVE 'DSN8ICDO' TO MODNAME
              ELSE MOVE 'DSN8ICGO' TO MODNAME.

      ****************************************************************
      *   CALL DL ISRT OUTPUT MESSAGE                               *
      *   PRINT ERROR MESSAGE IF STATUS CODE NOT OK                 *
      ****************************************************************

       CSEND.

      *                                         **CALL DL1 ISRT
           CALL 'CBLTDLI'
           USING ISRT-FKT IOPCB OUT-MESSAGE MODNAME.

      *                                         **STATUS CODE OK
           IF STC-CODE IN IOPCB = ' ' THEN GO TO CEND.

      *                                        **STATUS CODE NOT OK
      *                                        **PRINT ERROR MESSAGE
              MOVE '065E' TO MSGCODE.
              CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
              MOVE OUTMSG TO MSGTEXT IN OUTPUTAREA.

              MOVE STC-CODE IN IOPCB TO STC     IN OUTPUTAREA.

      *                                         **CALL DL1 CHNG
           CALL 'CBLTDLI'
           USING CHNG-FKT ALTPCB IOLTERM.

      *                                         **ERROR?
           IF STC-CODE IN ALTPCB NOT = ' ' THEN
               GO TO CSEND1.

      *                                         **CALL DL1 ISRT
           CALL 'CBLTDLI'
           USING ISRT-FKT IOPCB OUT-MESSAGE MODNAME.

      *                                         **PERFORM ROLLBACK
       CSEND1.
           CALL 'CBLTDLI' USING ROLL-FKT.

      *                                         **RETURN
       CEND.
           GOBACK.
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSN8IC1

THIS MODULE RETRIEVES THE ROW CONTAINING INFORMATION ON THE CURRENT CONVERSATION, VALIDATES SELECTION CRITERIA, AND ISSUES MESSAGES TO COMPLETE THE ACTION, OBJECT, AND SEARCH CRITERIA.

```
       IDENTIFICATION DIVISION.
      *------------------------
       PROGRAM-ID. DSN8IC1.

      *********** DSN8IC1 - SQL 1 MAINLINE FOR IMS - COBOL ************
      *                                                               *
      *   MODULE NAME = DSN8IC1                                       *
      *                                                               *
      *   DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                   *
      *                      SQL 1 MAINLINE                           *
      *                      IMS                                      *
      *                      COBOL                                    *
      *                                                               *
      *COPYRIGHT = 5615-DB2 (C) COPYRIGHT IBM CORP 1982, 2013         *
```

```
*REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083          *
*                                                               *
*STATUS = VERSION 11                                            *
*                                                               *
*   FUNCTION = THIS MODULE RETRIEVES THE ROW CONTAINING         *
*              INFORMATION ON THE CURRENT CONVERSATION,         *
*              VALIDATES SELECTION CRITERIA, AND ISSUES         *
*              MESSAGES TO COMPLETE THE ACTION, OBJECT,         *
*              AND SEARCH CRITERIA.                             *
*                                                               *
*   NOTES =  NONE                                               *
*                                                               *
*   MODULE TYPE =                                               *
*      PROCESSOR   = DB2  PRECOMPILER, COBOL COMPILER           *
*      MODULE SIZE = SEE LINKEDIT                               *
*      ATTRIBUTES  = REUSABLE                                   *
*                                                               *
*   ENTRY POINT = DSN8IC1                                       *
*      PURPOSE = SEE FUNCTION                                   *
*      LINKAGE = CALLED BY DSN8IC0                              *
*                                                               *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:   *
*                                                               *
*              SYMBOLIC LABEL/NAME = COMMPTR                    *
*              DESCRIPTION         = POINTER TO COMMAREA        *
*                                                               *
*              SYMBOLIC LABEL/NAME = INAREA                     *
*              DESCRIPTION         = USER INPUT                 *
*                                                               *
*              SYMBOLIC LABEL/NAME = PFKIN                      *
*              DESCRIPTION         = 00/01/02/03/07/08/10/11    *
*                                                               *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                 *
*                                                               *
*              SYMBOLIC LABEL/NAME = OUTAREA                    *
*              DESCRIPTION         = GENERAL MENU OR SECONDARY  *
*                                    SELECTION MENU             *
*                                                               *
*              SYMBOLIC LABEL/NAME =  LASTSCR                   *
*              DESCRIPTION = DSN8001/DSN8002                    *
*                                                               *
*   EXIT-NORMAL = DSN8IC0                                       *
*                                                               *
*   EXIT-ERROR = DSN8IC0                                        *
*                                                               *
*      RETURN CODE = NONE                                       *
*                                                               *
*      ABEND CODES =   NONE                                     *
*                                                               *
*      ERROR-MESSAGES =   NONE                                  *
*                                                               *
*   EXTERNAL REFERENCES =                                       *
*      ROUTINES/SERVICES =                                      *
*       DSN8IC2                                                 *
*       DSN8MCG                                                 *
*       DSNTIAR                                                 *
*                                                               *
*      DATA-AREAS =                                             *
*       DSN8MCCA          - COBOL STRUCTURE FOR COMMAREA        *
*       DSN8MCC2          - COMMAREA PART 2                     *
*       DSN8MCCS          - VCONA TABLE DCL & PCONA DCLGEN      *
*       DSN8MCOV          - VOPTVAL TABLE DCL & POPTVAL DCLGEN  *
*       DSN8MCVO          - VALIDATION CURSORS                  *
*       DSN8MCXX          - SQL ERROR HANDLING MODULE           *
*       DSN8MC1           - SQL1 COMMON MODULE FOR IMS & CICS   *
*       DSN8MC3 - DSN8MC5 - VALIDATION MODULES CALLED BY DSN8MC1*
*                                                               *
*      CONTROL-BLOCKS =                                         *
*       SQLCA             - SQL COMMUNICATION AREA              *
*                                                               *
*   TABLES = NONE                                               *
*                                                               *
*   CHANGE-ACTIVITY = NONE                                      *
*                                                               *
*                                                               *
*  *PSEUDOCODE*                                                 *
*     PROCEDURE                                                 *
*      INCLUDE DECLARATIONS.                                    *
*      INCLUDE DSN8MC1.                                         *
*                                                               *
*      CC1EXIT:  ( REFERENCED BY DSN8MC1 )                      *
*          RETURN.                                              *
```

```
*                                                                *
*      CC1CALL:  ( REFERENCED BY DSN8MC1 )                       *
*          CALL 'DSN8IC2' USING COMMAREA.                        *
*          GO TO MC1SAVE.  (LABEL IN DSN8MC1)                    *
*                                                                *
*      INCLUDE VALIDATION MODULES.                               *
*                                                                *
*     END.                                                       *
*----------------------------------------------------------------*
*
 ENVIRONMENT DIVISION.
*--------------------

 DATA DIVISION.
*--------------------
 WORKING-STORAGE SECTION.
*****************************************************************
*      * DECLARE FIELD SENT TO MESSAGE ROUTINE                  *
*      * DECLARE CONVERSATION STATUS                            *
*      * DECLARE MESSAGE TEXT                                   *
*      * DECLARE OPTION VALIDATION                              *
*      * DECLARE COMMON AREA AND COMMON AREA PART 2             *
*****************************************************************

 01  MSGCODE         PIC X(04).

 01  OUTMSG          PIC X(69).

     EXEC SQL INCLUDE DSN8MCCS END-EXEC.
     EXEC SQL INCLUDE DSN8MCOV END-EXEC.
     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
*
 LINKAGE SECTION.
 01  COMMAREA.
     EXEC SQL INCLUDE DSN8MCCA END-EXEC.
*
 PROCEDURE DIVISION USING COMMAREA.
*------------------

********************************************************
*           **SQL ERROR HANDLING
********************************************************
     EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
     EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

*
     MOVE 'DSN8IC1 ' TO MAJOR IN DSN8-MODULE-NAME.

***********************************************************
*  FIND VALID OPTIONS FOR ACTION, OBJECT, SEARCH CRITERION*
*  RETRIEVE CONVERSATION, VALIDATE, CALL SQL2            *
***********************************************************

     EXEC SQL INCLUDE DSN8MCVO END-EXEC.
*                                         **INCLUDE SQL1 MAIN
     EXEC SQL INCLUDE DSN8MC1 END-EXEC.
*
*                                         **RETURN
 CC1-EXIT.
     GOBACK.
***********************************************************
*  VALIDATE ACTION, OBJECT, SEARCH CRITERIA              *
*  HANDLE ERRORS                                         *
***********************************************************

 CC1-CALL.
     CALL 'DSN8IC2' USING COMMAREA.
     GO TO MC1-SAVE.

     EXEC SQL INCLUDE DSN8MC3 END-EXEC.
     EXEC SQL INCLUDE DSN8MC4 END-EXEC.
     EXEC SQL INCLUDE DSN8MC5 END-EXEC.

     EXEC SQL INCLUDE DSN8MCXX END-EXEC.
     GOBACK.
```

Related reference

"Sample applications in IMS" on page 1332

A set of Db2 sample applications run in the IMS environment.

DSN8IC2

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING CALLS SECONDARY SELECTION MODULES DSN8MCA DSN8MCM CALLS DETAIL MODULES DSN8MCD DSN8MCE DSN8MCF DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ CALLED BY DSN8IC1 (SQL1) .

```
        IDENTIFICATION DIVISION.                                        00010000
       *-----------------------                                         00020000
        PROGRAM-ID. DSN8IC2.                                            00030000
                                                                        00040000
       ******* DSN8IC2 - SQL 2 COMMON MODULE FOR IMS - COBOL ******     00050000
       *                                                                00060000
       *   MODULE NAME = DSN8IC2                                        00070000
       *                                                                00080000
       *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                   00090000
       *                      SQL 2 COMMON MODULE                       00100000
       *                      IMS                                       00110000
       *                      COBOL                                     00120000
       *                                                                00130000
       * LICENSED MATERIALS - PROPERTY OF IBM                           00132000
       * 5615-DB2                                                       00134000
       * (C) COPYRIGHT 1995, 2013 IBM CORP.  ALL RIGHTS RESERVED        00137000
       *                                                                00140000
       * STATUS = VERSION 11                                            00150000
       *                                                                00200000
       *                                                                00203000
       *                                                                00206000
       *   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR             00210000
       *                         DETAIL PROCESSING                      00220000
       *              CALLS SECONDARY SELECTION MODULES                 00230000
       *                    DSN8MCA DSN8MCM                             00240000
       *              CALLS DETAIL MODULES                              00250000
       *                    DSN8MCD DSN8MCE DSN8MCF                     00260000
       *                    DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ     00270000
       *              CALLED BY DSN8IC1 (SQL1)                          00280000
       *                                                                00290000
       *   NOTES = NONE                                                 00300000
       *                                                                00310000
       *                                                                00320000
       *   MODULE TYPE =                                                00330000
       *      PROCESSOR   = DB2  PRECOMPILER, COBOL COMPILER            00340000
       *      MODULE SIZE = SEE LINKEDIT                                00350000
       *      ATTRIBUTES  = REUSABLE                                    00360000
       *                                                                00370000
       *   ENTRY POINT = DSN8IC2                                        00380000
       *      PURPOSE = SEE FUNCTION                                    00390000
       *      LINKAGE = NONE                                            00400000
       *      INPUT = POINTER TO COMMAREA  (COMMUNICATION AREA)         00410000
       *                                                                00420000
       *              SYMBOLIC LABEL/NAME =  COMMAREA                   00430000
       *              DESCRIPTION = COMMUNICATION AREA PASSED BETWEEN   00440000
       *                            MODULES                             00450000
       *                                                                00460000
       *      OUTPUT = POINTER TO COMMAREA (COMMUNICATION AREA)         00470000
       *                                                                00480000
       *              SYMBOLIC LABEL/NAME =  COMMAREA                   00490000
       *              DESCRIPTION = COMMUNICATION AREA PASSED BETWEEN   00500000
       *                            MODULES                             00510000
       *                                                                00520000
       *   EXIT-NORMAL =                                                00530000
       *                                                                00540000
       *   EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER         00550000
       *                CONDITION SPECIFIED IN DSN8IC2 WILL BE RAISED   00560000
       *                AND PROGRAM WILL GO TO THE LABEL DB-ERROR.      00570000
       *                                                                00580000
       *                                                                00590000
       *      RETURN CODE = NONE                                        00600000
       *                                                                00610000
       *      ABEND CODES = NONE                                        00620000
       *                                                                00630000
       *      ERROR-MESSAGES =                                          00640000
       *         DSN8062E-AN OBJECT WAS NOT SELECTED                    00650000
       *         DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR                00660000
       *         DSN8072E-INVALID SELECTION ON SECONDARY SCREEN         00670000
       *                                                                00680000
       *                                                                00690000
       *   EXTERNAL REFERENCES =                                        00700000
```

```
      *      ROUTINES/SERVICES = 10 MODULES LISTED ABOVE
      *         DSN8MCG           -   ERROR MESSAGE ROUTINE
      *
      *      DATA-AREAS =
      *         DSN8MCA           -   SECONDARY SELECTION FOR
      *         DSN8MCD           -   DEPARTMENT STRUCTURE DETAIL
      *         DSN8MCE           -   DEPARTMENT DETAIL
      *         DSN8MCF           -   EMPLOYEE   DETAIL
      *                               ORGANIZATION
      *         DSN8MCAD          -   DECLARE ADMINISTRATION DETAIL
      *         DSN8MCAE          -   CURSOR EMPLOYEE LIST
      *         DSN8MCAL          -   CURSOR ADMINISTRATION LIST
      *         DSN8MCA2          -   DECLARE ADMINISTRATION DETAIL
      *         DSN8MCC2          -   SQL COMMON AREA PART 2
      *         DSN8MCDA          -   CURSOR ADMINISTRATION DETAIL
      *         DSN8MCDH          -   CURSOR FOR DISPLAY TEXT FROM
      *                               TDSPTXT TABLE
      *         DSN8MCDP          - DECLARE DEPARTMENT
      *         DSN8MCEM          - DECLARE EMPLOYEE
      *         DSN8MCED          - DECLARE EMPLOYEE-DEPARTMENT
      *         DSN8MCDM          - DECLARE DEPARTMENT MANAGER
      *         DSN8MCAD          - DECLARE ADMINISTRATION DETAIL
      *         DSN8MCA2          - DECLARE ADMINISTRATION DETAIL
      *         DSN8MCOV          - DECLARE OPTION VALIDATION
      *         DSN8MCDT          - DECLARE DISPLAY TEXT
      *         DSN8MCCA          - SQL COMMON AREA
      *         DSN8MCXX          - ERROR HANDLER
      *
      *      CONTROL-BLOCKS =
      *         SQLCA             - SQL COMMUNICATION AREA
      *
      *   TABLES = NONE
      *
      *   CHANGE-ACTIVITY =
      *   - ADD NEW VARIABLES FOR REFERENTIAL INTEGRITY         V2R1
      *
      *  *PSEUDOCODE*
      *
      * THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR
      * DETAIL MODULE(S) ARE TO BE CALLED FOR THE IMS/COBOL ENVIRONMENT
      *
      * WHAT HAS HAPPENED SO FAR?............. THE SUBSYSTEM
      * DEPENDENT MODULE (IMS,CICS) (SQL 0) HAS READ THE
      * INPUT SCREEN, FORMATTED THE INPUT, AND PASSED CONTROL
      * TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT
      * FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF
      * ALL SYSTEM FIELDS ARE VALID, SQL 1 PASSED CONTROL TO THIS
      * MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH
      * POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE
      * BETWEEN SQL 0 , SQL 1, SQL 2, AND THE SECONDARY SELECTION
      * AND DETAIL MODULES.
      *
      * WHAT IS INCLUDED IN THIS MODULE?...........
      * ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.
      * ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE
      * SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND
      * SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. ALL CURSOR HOST
      * VARIABLES ARE DECLARED IN THIS PROCEDURE BECAUSE OF THE
      * RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE
      * THE CURSOR DEFINITION.
      *
      * PROCEDURE
      *    IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR
      *    IS NOT WILLING TO ACCEPT AN ANSWER THEN
      *                    NEW REQUEST*
      *
      *    ELSE
      *           IF ANSWER TO A SECONDARY SELECTION THEN
      *                    DETERMINE IF NEW REQUEST.
      *
      *    CASE (NEW REQUEST)
      *
      *           SUBCASE ('ADD')
      *                   DETAIL PROCESSOR
      *                   RETURN TO SQL 1
      *           ENDSUB
      *
      *           SUBCASE ('DISPLAY','ERASE','UPDATE')
      *                   CALL SECONDARY SELECTION
      *                   IF # OF POSSIBLE CHOICES IS ^= 1 THEN
      *                          RETURN TO SQL 1
      *                   ELSE
```

```
*                          CALL THE DETAIL PROCESSOR
*                          RETURN TO SQL 1.
*           ENDSUB
*
*    ENDCASE
*
*    IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS
*    ACTUALLY BEEN MADE THEN
*                VALID SELECTION #?
*                IF IT IS VALID THEN
*                      CALL DETAIL PROCESSOR
*                      RETURN TO SQL 1
*                ELSE
*                      PRINT ERROR MSG
*                      RETURN TO SQL 1.
*
*    IF ANSWER TO SECONDARY SELECTION THEN
*             CALL SECONDARY SELECTION
*             RETURN TO SQL 1.
*
*    IF ANSWER TO DETAIL THEN
*             CALL DETAIL PROCESSOR
*             RETURN TO SQL 1.
*
*  END.
*
*   *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES
*    THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.
*-----------------------------------------------------------
/

 ENVIRONMENT DIVISION.
*--------------------------

 DATA DIVISION.
*-------------------
 WORKING-STORAGE SECTION.

*******************************************************
*   FIELD  SENT TO MESSAGE ROUTINE
*******************************************************
  01 MSGCODE                 PIC X(04).
  01 OUTMSG                  PIC X(69).

***************************************
*  NULL INDICATOR                     *
***************************************
  01 NULLIND1                PIC S9(4) COMP-4.
  01 NULLIND2                PIC S9(4) COMP-4.
  01 NULLIND3                PIC S9(4) COMP-4.
  01 NULLIND4                PIC S9(4) COMP-4.
  01 NULLIND5                PIC S9(4) COMP-4.
  01 NULLARRY.
     03  NULLARRY1  PIC S9(4) USAGE COMP OCCURS 13 TIMES.

     EXEC SQL INCLUDE SQLCA END-EXEC.

     EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
     EXEC SQL INCLUDE DSN8MCDP END-EXEC.
     EXEC SQL INCLUDE DSN8MCEM END-EXEC.
     EXEC SQL INCLUDE DSN8MCDM END-EXEC.
     EXEC SQL INCLUDE DSN8MCAD END-EXEC.
     EXEC SQL INCLUDE DSN8MCA2 END-EXEC.
     EXEC SQL INCLUDE DSN8MCOV END-EXEC.
     EXEC SQL INCLUDE DSN8MCDT END-EXEC.
     EXEC SQL INCLUDE DSN8MCED END-EXEC.

  01  CONSTRAINTS.
      03  PARM-LENGTH        PIC S9(4) COMP-4.
      03  REF-CONSTRAINT     PIC X(08).
      03  FILLER             PIC X(62).
  01  MGRNO-CONSTRAINT       PIC X(08)  VALUE 'RDE     '.

 LINKAGE SECTION.
 01  COMMAREA.
     EXEC SQL INCLUDE DSN8MCCA END-EXEC.

 PROCEDURE DIVISION USING COMMAREA.
*------------------
*******************************************************
*   SQL ERROR CODE HANDLING
*******************************************************
```

```
           EXEC SQL WHENEVER SQLERROR   GO TO DB-ERROR END-EXEC
           EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

           EXEC SQL INCLUDE DSN8MCAE END-EXEC.
           EXEC SQL INCLUDE DSN8MCAL END-EXEC.
           EXEC SQL INCLUDE DSN8MCDH END-EXEC.
           EXEC SQL INCLUDE DSN8MCDA END-EXEC.

      *******************************************************
      *   INITIALIZATIONS
      *******************************************************

           MOVE 'DSN8IC2' TO MAJOR.
           MOVE  SPACES   TO MINOR.


           IF NEWREQ OF COMPARM = 'Y' THEN GO TO IC2008.

      *******************************************************
      *   DETERMINES WHETHER NEW REQUEST OR NOT
      *******************************************************
       IC2005.
           IF PREV OF PCONVSTA = ' ' THEN
              MOVE 'Y' TO NEWREQ OF COMPARM.

           IF NEWREQ OF COMPARM = 'N' AND PREV OF PCONVSTA = 'S'
              AND DATA01 NOT = ' '
              AND DATAIN NOT = 'NEXT'
              THEN MOVE 'Y' TO NEWREQ OF COMPARM.

           IF NEWREQ OF COMPARM NOT = 'Y' THEN GO TO IC2010.
      *******************************************************
      *   IF NEW REQUEST AND ACTION IS 'ADD' THEN
      *        CALL DETAIL PROCESSOR
      *   ELSE CALL SECONDARY SELECTION
      *******************************************************
       IC2008.
           IF ACTION OF INAREA = 'A' THEN
      *                                     **DETAIL PROCESSOR
              GO TO DETAIL0.
      *                                     **SECONDARY SELECTION
           PERFORM SECSEL THRU END-SECSEL.
      *                                     **IF NO. OF CHOICES = 1
      *                                     **GO TO DETAIL PROCESSOR
           IF MAXSEL = 1 THEN GO TO DETAIL0.
           GO TO EXIT0.
      *******************************************************
      *   DETERMINE IF VALID SELECTION NUMBER GIVEN
      *******************************************************
       IC2010.
      *                                     **VALID SELECTION NO. GIVEN
           IF PREV OF PCONVSTA NOT = 'S'
              OR MAXSEL < 1
              OR DATAIN = 'NEXT'
              OR DATA2 = DATO2  THEN GO TO IC201.
      *
           IF DAT1 NUMERIC AND DAT2 = ' ' THEN
              MOVE DAT1 TO DAT2
              MOVE '0' TO DAT1.

      *                                        **DETAIL SELECTION GIVEN
           IF DATA2 NUMERIC
              AND DATA2 > '00' AND DATA2 NOT > MAXSEL THEN
              MOVE 'Y' TO NEWREQ OF COMPARM
              GO TO DETAIL0.

      *                                   **INVALID SELECTION NO.
      *                                   **PRINT ERROR MESSAGE
           MOVE '072E' TO MSGCODE.
           CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
           MOVE OUTMSG TO MSG OF OUTAREA.
           GO TO EXIT0.

      *******************************************************
      *  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL
      *******************************************************
       IC201.
      *                                           **SECONDARY SELECTION
           IF PREV OF PCONVSTA = 'S' THEN
              PERFORM SECSEL THRU END-SECSEL
              GO TO EXIT0.

```

```
*                                             **DETAIL PROCESSOR
     IF PREV OF PCONVSTA = 'D' THEN GO TO DETAIL0.

*                                           **LOGIC ERROR
*                                           **PRINT ERROR MESSAGE
     MOVE '066E' TO MSGCODE.
     CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
     MOVE OUTMSG TO MSG OF OUTAREA.
     GO TO EXIT0.

***********************************************************
*  CALLS SECONDARY SELECTION PROCESSOR AND RETURNS TO SQL 1
***********************************************************
 SECSEL.
     MOVE 'DSN8001 ' TO LASTSCR IN PCONVSTA.
     IF OBJFLD OF INAREA = 'DS' THEN
*                                          **ADMINISTRATIVE
*                                          **DEPARTMENT STRUCTURE
        PERFORM DSN8MCA THRU END-DSN8MCA
     ELSE
        IF OBJFLD OF INAREA = 'DE' THEN
*                                         **INDIVIDUAL DEPARTMENT
*                                         **PROCESSING
           PERFORM DSN8MCA THRU END-DSN8MCA
        ELSE
           IF OBJFLD OF INAREA = 'EM' THEN
*                                          **INDIVIDUAL EMPLOYEE
*                                          **PROCESSING
              PERFORM DSN8MCA THRU END-DSN8MCA
           ELSE
*                                        **ERROR MESSAGE
*                                        **UNSUPPORTED SEARCH
*                                        **CRITERIA FOR OBJECT
             MOVE '062E' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO MSG OF OUTAREA
             GO TO EXIT0.
 END-SECSEL.

*******************************************************
*   CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1
*******************************************************
 DETAIL0.
     MOVE 'DSN8002 ' TO LASTSCR IN PCONVSTA.

     IF OBJFLD OF INAREA = 'DS' THEN
*                                          **ADMINISTRATIVE
*                                          **DEPARTMENT STRUCTURE
        PERFORM DSN8MCD THRU END-DSN8MCD
     ELSE
        IF OBJFLD OF INAREA = 'DE' THEN
*                                         **INDIVIDUAL DEPARTMENT
*                                         **PROCESSING
           PERFORM DSN8MCE THRU END-DSN8MCE
        ELSE
           IF OBJFLD OF INAREA = 'EM' THEN
*                                          **INDIVIDUAL EMPLOYEE
*                                          **PROCESSING
              PERFORM DSN8MCF THRU END-DSN8MCF
           ELSE
*                                        **ERROR MESSAGE
*                                        **UNSUPPORTED SEARCH
*                                        **CRITERIA FOR OBJECT
             MOVE '062E' TO MSGCODE
             CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
             MOVE OUTMSG TO MSG OF OUTAREA.
     GO TO EXIT0.
*                                        **HANDLES ERRORS
*                                        **RETURN TO SQL 1
     EXEC SQL INCLUDE DSN8MCXX END-EXEC.
 EXIT0. GOBACK.

     EXEC SQL INCLUDE DSN8MCA END-EXEC.
     EXEC SQL INCLUDE DSN8MCD END-EXEC.
     EXEC SQL INCLUDE DSN8MCE END-EXEC.
     EXEC SQL INCLUDE DSN8MCF END-EXEC.
```

Related reference

"Sample applications in IMS" on page 1332

A set of Db2 sample applications run in the IMS environment.

DSN8IP0

THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT, CALLS DSN8IP1, FORMATS OUTPUT MESSAGE AND SENDS IT .

```
DSN8IP0: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN);
/*******************************************************************
*                                                                  *
*   MODULE NAME = DSN8IP0                                          *
*                                                                  *
*   DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                      *
*                      SUBSYSTEM INTERFACE MODULE                  *
*                      IMS                                         *
*                      PL/I                                        *
*                      ORGANIZATION APPLICATION                    *
*                                                                  *
*    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985        *
*    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083         *
*                                                                  *
*    STATUS = RELEASE 2, LEVEL 0                                   *
*                                                                  *
*   FUNCTION = THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT, *
*              CALLS DSN8IP1, FORMATS OUTPUT MESSAGE AND SENDS IT  *
*                                                                  *
*   NOTES =    NONE                                                *
*                                                                  *
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                          *
*      PROCESSOR   = PL/I OPTIMIZER                                *
*      MODULE SIZE = SEE LINKEDIT                                  *
*      ATTRIBUTES  = REUSABLE                                      *
*                                                                  *
*   ENTRY POINT =  DSN8IP0                                         *
*      PURPOSE = SEE FUNCTION                                      *
*      LINKAGE = FROM IMS                                          *
*                                                                  *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:      *
*                                                                  *
*         SYMBOLIC LABEL/NAME  =  DSN8IPGI                         *
*         DESCRIPTION = IMS/VS MFS GENERAL MENU                    *
*                                                                  *
*         SYMBOLIC LABEL/NAME  =  DSN8IPDI                         *
*         DESCRIPTION = IMS/VS MFS SECONDARY SELECTION MENU        *
*                                                                  *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                    *
*                                                                  *
*         SYMBOLIC LABEL/NAME  =  DSN8IPGO                         *
*         DESCRIPTION = IMS/VS MFS GENERAL MENU                    *
*                                                                  *
*         SYMBOLIC LABEL/NAME  =  DSN8IPDO                         *
*         DESCRIPTION = IMS/VS MFS SECONDARY SELECTION MENU        *
*                                                                  *
*   EXIT-NORMAL =                                                  *
*                                                                  *
*   EXIT-ERROR =                                                   *
*                                                                  *
*      RETURN CODE = NONE                                          *
*                                                                  *
*      ABEND CODES =  NONE                                         *
*                                                                  *
*      ERROR-MESSAGES =                                            *
*          DSN8064E - INVALID DL/I STC-CODE ON GU MSG              *
*          DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG            *
*                                                                  *
*   EXTERNAL REFERENCES =                                          *
*      ROUTINES/SERVICES =  MODULE DSN8IP1                         *
*                           MODULE PLITDLI                         *
*                           MODULE DSN8MPG                         *
*                                                                  *
*      DATA-AREAS =                                                *
*         DSN8MPCA            - PARAMETER TO BE PASSED TO DSN8CP1  *
*                               CONTAINS TERMINAL INPUT AND        *
*                               OUTPUT AREAS.                      *
*         IN_MESSAGE          - MFS INPUT                          *
*         OUT_MESSAGE         - MFS OUTPUT                         *
*                                                                  *
*      CONTROL-BLOCKS = NONE                                       *
*                                                                  *
```

```
 *   TABLES = NONE                                                    *
 *                                                                    *
 *   CHANGE-ACTIVITY = NONE                                           *
 *                                                                    *
 *                                                                    *
 *  *PSEUDOCODE*                                                      *
 *                                                                    *
 *      PROCEDURE                                                     *
 *             DECLARATIONS.                                          *
 *             ALLOCATE PL/I WORK AREA FOR COMMAREA.                  *
 *             INITIALIZATION.                                        *
 *                 PUT MODNAME 'DSN8IPGO' IN MODNAME FIELD.           *
 *                 PUT MODULE NAME 'DSN8IP0' IN AREA USED BY          *
 *                      ERROR_HANDLER.                                *
 *                                                                    *
 *           STEP1.                                                   *
 *           CALL DLI GU INPUT MESSAGE.                               *
 *           IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND        *
 *              STOP PROGRAM.                                         *
 *                                                                    *
 *           IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.      *
 *              MOVE INPUT MESSAGE FIELDS TO CORRESPONDING            *
 *                   INAREA FIELDS IN COMPARM.                        *
 *              CALL DSN8IP1 (COMMAREA)                               *
 *              MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING      *
 *                   OUTPUT MESSAGE FIELDS.                           *
 *              IF LASTSCR 'DSN8001' MOVE 'DSN8IPGO' TO MODNAME FIELD *
 *                   ELSE MOVE 'DSN8IPDO' TO MODNAME FIELD.           *
 *                                                                    *
 *           CALL DLI ISRT OUTPUT MESSAGE.                            *
 *           IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND        *
 *              STOP PROGRAM.                                         *
 *      END.                                                          *
 *                                                                    *
 *--------------------------------------------------------------------*
1/********************************************************************/
 /*       DECLARATION FOR INPUT:  MIDNAME DSN8IPGI/DSN8IPDI          */
 /********************************************************************/
0DCL  1 IN_MESSAGE     STATIC,
       2 LL             BIN FIXED (31),
       2 Z1             CHAR (1),
       2 Z2             CHAR (1),
       2 TC_CODE        CHAR (7),
       2 MESSAGE,
         3 INPUT,
           5 MAJSYS       CHAR (1),
           5 ACTION       CHAR (1),
           5 OBJFLD       CHAR (2),
           5 SEARCH       CHAR (2),
           5 PFKIN        CHAR (2),
           5 DATA         CHAR (60),
           5 TRANDATA(15) CHAR (40);
-/********************************************************************/
 /*        DECLARATION FOR OUTPUT: MODNAME DSN8IPGO/DSN8IPDO         */
 /********************************************************************/
0DCL  1 OUT_MESSAGE    STATIC,
       2 LL           BIN FIXED (31) INIT (1613),
       2 ZZ           BIN FIXED (15) INIT (0),
       2 OUTPUT,
         3 OUTPUTAREA,
           5 MAJSYS       CHAR (1),
           5 ACTION       CHAR (1),
           5 OBJFLD       CHAR (2),
           5 SEARCH       CHAR (2),
           5 DATA         CHAR (60),
           5 TITLE        CHAR (50),
           5 DESC2        CHAR (50),
           5 DESC3        CHAR (50),
           5 DESC4        CHAR (50),
           5 MSG          CHAR (79),
           5 PFKTEXT      CHAR (79),
           5 OUTPUT,
             7 LINE (15)  CHAR (79);
1/********************************************************************/
 /*        DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE     */
 /*        SUBSYSTEM DEPENDENT MODULE IMS/DL1 AND SQL1 AND SQL2      */
 /********************************************************************/
      EXEC SQL INCLUDE DSN8MPCA;

  DCL DSN8MPG EXTERNAL ENTRY;

1/********************************************************************/
```

```
 /*        FIELDS SENT TO MESSAGE ROUTINE                                  */
 /*******************************************************************/
  DCL MODULE              CHAR(07) INIT('DSN8IP0');
  DCL OUTMSG              CHAR(69);
1/*******************************************************************/
 /*        DECLARATION FOR PGM-LOGIC                                       */
 /*******************************************************************/
0DCL  ONE        BIN FIXED (31) INIT (1)  STATIC;
 DCL  THREE      BIN FIXED (31) INIT (3)  STATIC;
 DCL  FOUR       BIN FIXED (31) INIT (4)  STATIC;
0DCL  GU_FKT     CHAR (4) INIT ('GU  ') STATIC;
 DCL  ISRT_FKT   CHAR (4) INIT ('ISRT') STATIC;
 DCL  CHNG_FKT   CHAR (4) INIT ('CHNG') STATIC;
 DCL  ROLL_FKT   CHAR (4) INIT ('ROLL') STATIC;
0DCL  MODNAME    CHAR (8) STATIC;
0DCL  (ADDR,LOW) BUILTIN;
0DCL  PLITDLI EXTERNAL ENTRY;
 DCL  DSN8IP1 EXTERNAL ENTRY;
0DCL  (IOPCB_ADDR,ALTPCB_ADDR) POINTER;
1/*******************************************************************/
 /*        DECLARATION FOR IO / ALTPCB MASK                                */
 /*******************************************************************/
0DCL  1 IOPCB        BASED (IOPCB_ADDR),
        2 IOLTERM    CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2),
        2 CDATE      CHAR (4),
        2 CTIME      CHAR (4),
        2 SEQNUM     CHAR (4),
        2 MOD_NAME   CHAR (8),
        2 USERID     CHAR (8);
0DCL  1 ALTPCB       BASED (ALTPCB_ADDR),
        2 ALTLTERM   CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2);
     /***************************************************************/
     /*         ALLOCATE COBOL WORK AREA /INITIALIZATIONS           */
     /***************************************************************/

    ALLOCATE COMMAREA SET(COMMPTR);
         COMMAREA   = '';                      /* CLEAR COMMON AREA*/
         IN_MESSAGE = '';                      /* CLEAR INPUT FIELD*/
         MODNAME                = 'DSN8IPGO';  /* GET MODULE  NAME */
         DSN8_MODULE_NAME.MAJOR = 'DSN8IP0';   /* GET MODULE NAME  */
         OUTAREA.MAJSYS = 'O';                 /* MAJOR SYSTEM - O */
         EXITCODE = '0';                       /* CLEAR EXIT CODE  */

     /***************************************************************/
     /*     CALL DL1 GU INPUT MESSAGE                               */
     /*     PRINT ERROR MESSAGE IF STATUS CODE NOT OK               */
     /***************************************************************/

0   CALL PLITDLI (THREE,GU_FKT,IOPCB,IN_MESSAGE);  /*CALL DL1 GU */

0   IF   IOPCB.STC_CODE ^= ' ' THEN                /* ERROR ?    */
         DO;
          CALL DSN8MPG (MODULE, '064E', OUTMSG);
          OUTPUTAREA.MSG = OUTMSG;


          GO TO CSEND;             /*CALL DL1 ISRT OUTPUT MESSAGE */
         END;

     /***************************************************************/
     /*     CLEARED AND UNFORMATTED SCREEN?                         */
     /***************************************************************/

    IF   Z2 = LOW(1) THEN COMPARM.PFKIN   = '00';
0        PCONVSTA.CONVID  = IOPCB.IOLTERM||USERID;

         INAREA           = INPUT, BY NAME;     /*MOVE INPUT MESSAGE */
         INAREA.MAJSYS    = 'O';            /*FIELDS TO INAREA FIELDS*/

0   CALL DSN8IP1 (COMMPTR);

                                           /*MOVE OUTAREA FIELDS */
0        OUTPUTAREA = OUTAREA , BY NAME;   /*TO OUTPUT MESSAGE FIELDS*/
0   IF   LASTSCR = 'DSN8002' THEN MODNAME = 'DSN8IPDO';
                             ELSE MODNAME = 'DSN8IPGO';

     /***************************************************************/
     /*   CALL DL ISRT OUTPUT MESSAGE                               */
```

```
     /*   PRINT ERROR MESSAGE IF STATUS CODE NOT OK                 */
     /***************************************************************/

 CSEND:
                                               /*CALL DL1 ISRT*/
    CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME);

0   IF   IOPCB.STC_CODE = ' ' THEN GO TO CEND;     /*STATUS CODE OK*/

                                            /*STATUS CODE NOT OK*/
         CALL DSN8MPG (MODULE, '065E', OUTMSG);
0        OUTPUTAREA.MSG = OUTMSG;           /*PRINT ERROR MESSAGE*/

0   CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM); /*  CALL DL1 CHNG */

0   IF   ALTPCB.STC_CODE ^= ' ' THEN GO TO CSEND1; /* ERROR?       */

                                            /*   CALL DL1 ISRT   */
0   CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);

0CSEND1:                                    /*   PERFORM ROLLBACK*/
    CALL PLITDLI (ONE,ROLL_FKT);

0CEND:                                      /*   RETURN          */
    END DSN8IP0;
```

Related reference

[“Sample applications in IMS” on page 1332]
A set of Db2 sample applications run in the IMS environment.

DSN8IP1

PERFORM INCLUDES TO BRING IN SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS PARAMETER AREA.

```
 DSN8IP1:PROC (COMMPTR) ;
 /*********************************************************************
 *                                                                    *
 *   MODULE NAME = DSN8IP1                                            *
 *                                                                    *
 *   DESCRIPTIVE NAME = SAMPLE APPLICATION                            *
 *                      SQL 1 MAINLINE                                *
 *                      IMS                                           *
 *                      PL/I                                          *
 *                                                                    *
 *    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985          *
 *    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083           *
 *                                                                    *
 *    STATUS = RELEASE 2, LEVEL 0                                     *
 *                                                                    *
 *   FUNCTION = PERFORM INCLUDES TO BRING IN SQL TABLE DCLS AND       *
 *              DCLGEN STRUCTURES AS WELL AS PARAMETER AREA.          *
 *              INCLUDE DSN8MP1.                                      *
 *              CALL DSN8IP2.                                         *
 *              RETURN TO DSN8IP0.                                    *
 *                                                                    *
 *   NOTES =    NONE                                                  *
 *                                                                    *
 *   MODULE TYPE = PL/I PROC(COMMPTR).                                *
 *      PROCESSOR   = DB2  PRECOMPILER,  PL/I OPTIMIZER               *
 *      MODULE SIZE = SEE LINKEDIT                                    *
 *      ATTRIBUTES  = REUSABLE                                        *
 *                                                                    *
 *   ENTRY POINT = DSN8IP1                                            *
 *      PURPOSE = SEE FUNCTION                                        *
 *      LINKAGE = CALLED BY DSN8IP0                                   *
 *                                                                    *
 *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:        *
 *                                                                    *
 *              SYMBOLIC LABEL/NAME = COMMPTR                         *
 *              DESCRIPTION         = POINTER TO COMMUNICATION AREA   *
 *                                                                    *
 *              COMMON AREA.                                          *
 *                                                                    *
 *              SYMBOLIC LABEL/NAME = PFKIN                           *
 *              DESCRIPTION         = 00/01/02/03/08/10               *
 *                                                                    *
 *              SYMBOLIC LABEL/NAME = INAREA                          *
```

```
 *              DESCRIPTION         = USER INPUT                      *
 *                                                                   *
 *      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                     *
 *               COMMON AREA.                                        *
 *                                                                   *
 *               SYMBOLIC LABEL/NAME =  OUTAREA                      *
 *               DESCRIPTION         = GENERAL MENU OR SECONDARY     *
 *                                     SELECTION MENU                *
 *                                                                   *
 *               SYMBOLIC LABEL/NAME =  LASTSCR                      *
 *               DESCRIPTION         = DSN8001/DSN8002               *
 *                                                                   *
 *   EXIT-NORMAL = DSN8IP0                                           *
 *                                                                   *
 *   EXIT-ERROR = DSN8IP0                                            *
 *                                                                   *
 *      RETURN CODE = NONE                                           *
 *                                                                   *
 *      ABEND CODES = NONE                                           *
 *                                                                   *
 *      ERROR-MESSAGES = NONE                                        *
 *                                                                   *
 *   EXTERNAL REFERENCES =                                           *
 *      ROUTINES/SERVICES =  NONE                                    *
 *                                                                   *
 *      DATA-AREAS =                                                 *
 *         DSN8MPCA            - PLI STRUCTURE FOR COMMAREA          *
 *         DSN8MPCS            - VCONA TABLE DCL AND PCONA DCLGEN    *
 *         DSN8MPOV            - VOPTVAL TABLE DCL & POPTVAL DCLGEN  *
 *         DSN8MPVO            - VALIDATION CURSORS                  *
 *         DSN8MP1             - SQL1 COMMON MODULE FOR IMS AND CICS *
 *         DSN8MP3 -- DSN8MP5  - VALIDATION MODULES CALLED BY DSN8MP1 *
 *         DSN8MPXX            - SQL ERROR HANDLER                   *
 *                                                                   *
 *      CONTROL-BLOCKS =                                             *
 *         SQLCA               - SQL COMMUNICATION AREA              *
 *                                                                   *
 *   TABLES = NONE                                                   *
 *                                                                   *
 *   CHANGE-ACTIVITY = NONE                                          *
 *                                                                   *
 *                                                                   *
 *  *PSEUDOCODE*                                                     *
 *                                                                   *
 *    PROCEDURE                                                      *
 *      INCLUDE DECLARATIONS.                                        *
 *      INCLUDE DSN8MP1.                                             *
 *      INCLUDE ERROR HANDLER.                                       *
 *                                                                   *
 *      CP1EXIT:  ( REFERENCED BY DSN8MP1 )                          *
 *          RETURN.                                                  *
 *                                                                   *
 *      CP1CALL:  ( REFERENCED BY DSN8MP1 )                          *
 *          CALL 'DSN8IP2'(COMMPTR).                                 *
 *          GO TO MP1SAVE.  (LABEL IN DSN8MP1)                       *
 *                                                                   *
 *      INCLUDE VALIDATION MODULES.                                  *
 *                                                                   *
 *     END.                                                          *
 *********************************************************************/
1/********************************************************************/
 /*                           SQL1   MAINLINE                        */
 /*******************************************************************/
 DCL STRING BUILTIN;
 DCL J FIXED BIN;
 DCL SAVE_CONVID CHAR(16);
 DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);
                                        /* SQL RETURN CODE HANDLING */
 EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
 EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

 /*****************************************************/
 /*  FIELDS PASSED TO MESSAGE ROUTINE                 */
 /*****************************************************/
 DCL MODULE       CHAR(07);
 DCL OUTMSG       CHAR(69);

 DCL DSN8IP2 EXTERNAL ENTRY;
 DCL DSN8MPG EXTERNAL ENTRY;
 EXEC SQL INCLUDE DSN8MPCA;                /* INCLUDE COMMAREA */
 DSN8_MODULE_NAME.MAJOR = 'DSN8IP1';       /* INITIALIZE MODULE NAME*/
 EXEC SQL INCLUDE DSN8MPCS;                /* INCLUDE PCONA    */
```

```
EXEC SQL INCLUDE DSN8MPOV;                 /* INCLUDE POPTVAL  */
EXEC SQL INCLUDE DSN8MPVO;                 /* INCLUDE CURSOR   */
EXEC SQL INCLUDE SQLCA;                    /* SQL COMMON AREA  */
EXEC SQL INCLUDE DSN8MP1;                  /* INCLUDE SQL1 MAIN*/
EXEC SQL INCLUDE DSN8MPXX;                 /* HANDLES ERRORS   */

CP1EXIT :                                  /* STANDARD EXIT    */
     RETURN;

CP1CALL :
                                       /* GO TO DSN8IP2 (SQL2)     */
     CALL DSN8IP2 (COMMPTR);
     GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3;              /* INCLUDE ACTION VALIDATION*/
EXEC SQL INCLUDE DSN8MP4;              /* INCLUDE OBJECT VALIDATION*/
EXEC SQL INCLUDE DSN8MP5;              /* INCLUDE SEARCH CRITERIA*/
END;                                   /* VALIDATION             */
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSN8IP2

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN CALLS SECONDARY SELECTION MODULES DSN8MPA CALLS DETAIL MODULES DSN8MPD DSN8MPE DSN8MPF CALLED BY DSN8IP1 (SQL1) .

```
   DSN8IP2: PROC(COMMPTR) ;  /* SQL 2 FOR IMS AND PLI */              00010000
                                                              %PAGE;   00020000
 /********************************************************************* 00030000
 *                                                                   * 00040000
 *   MODULE NAME = DSN8IP2                                           * 00050000
 *                                                                   * 00060000
 *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                      * 00070000
 *                      SQL 2 COMMON MODULE                          * 00080000
 *                      IMS                                          * 00090000
 *                      PL/I                                         * 00100000
 *                      ORGANIZATION APPLICATION                     * 00110000
 *                                                                   * 00120000
 *     LICENSED MATERIALS - PROPERTY OF IBM                          * 00130000
 *     5695-DB2                                                      * 00136000
 *     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.      * 00143000
 *                                                                   * 00150000
 *     STATUS = VERSION 4                                            * 00160000
 *                                                                   * 00170000
 *   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN* 00180000
 *              CALLS SECONDARY SELECTION MODULES                    * 00190000
 *                    DSN8MPA                                        * 00200000
 *              CALLS DETAIL MODULES                                 * 00210000
 *                    DSN8MPD DSN8MPE DSN8MPF                        * 00220000
 *              CALLED BY DSN8IP1 (SQL1)                             * 00230000
 *                                                                   * 00240000
 *   NOTES =   NONE                                                  * 00250000
 *                                                                   * 00260000
 *                                                                   * 00270000
 *   MODULE TYPE = BLOCK OF PL/I CODE                                * 00280000
 *      PROCESSOR   = DB2  PRECOMPILER, PL/I OPTIMIZER               * 00290000
 *      MODULE SIZE = SEE LINKEDIT                                   * 00300000
 *      ATTRIBUTES  = REUSABLE                                       * 00310000
 *                                                                   * 00320000
 *   ENTRY POINT = DSN8IP2                                           * 00330000
 *      PURPOSE = SEE FUNCTION                                       * 00340000
 *      LINKAGE = CALL DSN8IP2(COMMPTR)                              * 00350000
 *      INPUT   =                                                    * 00360000
 *                                                                   * 00370000
 *               SYMBOLIC LABEL/NAME = COMMPTR                       * 00380000
 *               DESCRIPTION         = POINTER TO COMMUNICATION AREA * 00390000
 *                                                                   * 00400000
 *      OUTPUT  =                                                    * 00410000
 *                                                                   * 00420000
 *               SYMBOLIC LABEL/NAME = COMMPTR                       * 00430000
 *               DESCRIPTION         = POINTER TO COMMUNICATION AREA * 00440000
 *                                                                   * 00450000
 *   EXIT-NORMAL =                                                   * 00460000
 *                                                                   * 00470000
```

```
*   EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION   * 00480000
*                SPECIFIED IN DSN8IP2 WILL BE RAISED AND PROGRAM     * 00490000
*                WILL GO TO THE LABEL DB_ERROR.                      * 00500000
*                                                                    * 00510000
*                                                                    * 00520000
*      RETURN CODE = NONE                                            * 00530000
*                                                                    * 00540000
*      ABEND CODES = NONE                                            * 00550000
*                                                                    * 00560000
*      ERROR-MESSAGES =                                              * 00570000
*              DSN8062E-AN OBJECT WAS NOT SELECTED                   * 00580000
*              DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR               * 00600000
*              DSN8072E-INVALID SELECTION ON SECONDARY SCREEN        * 00610000
*                                                                    * 00620000
*   EXTERNAL REFERENCES =                                            * 00630000
*                                                                    * 00640000
*      ROUTINES/SERVICES =   MODULES LISTED ABOVE                    * 00650000
*         DSN8MPG             - ERROR MESSAGE ROUTINE                * 00660000
*                                                                    * 00670000
*      DATA-AREAS =                                                  * 00680000
*         DSN8MPA             - SECONDARY SELECTION FOR ORGANIZATION * 00690000
*         DSN8MPAD            - DECLARE ADMINISTRATIVE DETAIL        * 00700000
*         DSN8MPAE            - CURSOR EMPLOYEE LIST                 * 00710000
*         DSN8MPAL            - CURSOR ADMINISTRATION LIST           * 00720000
*         DSN8MPA2            - DECLARE ADMINISTRATIVE DETAIL        * 00730000
*         DSN8MPCA            - DECLARE SQL COMMON AREA              * 00740000
*         DSN8MPD             - DEPARTMENT STRUCTURE DETAIL          * 00750000
*         DSN8MPDA            - CURSOR ADMINISTRATION DETAIL         * 00760000
*         DSN8MPDH            - CURSOR FOR DISPLAY TEXT FROM         * 00770000
*                               TDSPTXT TABLE                        * 00780000
*         DSN8MPDM            - DECLARE DEPARTMENT MANAGER           * 00790000
*         DSN8MPDP            - DELCLARE DEPARTMENT                  * 00800000
*         DSN8MPDT            - DECLARE DISPLAY TEXT                 * 00810000
*         DSN8MPE             - DEPARTMENT DETAIL                    * 00820000
*         DSN8MPEM            - DECLARE EMPLOYEE                     * 00830000
*         DSN8MPED            - DECLARE EMPLOYEE-DEPARTMENT          * 00835000
*         DSN8MPF             - EMPLOYEE DETAIL                      * 00840000
*         DSN8MPOV            - DECLARE OPTION VALIDATION            * 00850000
*                                                                    * 00860000
*      CONTROL-BLOCKS =                                              * 00870000
*         SQLCA               - SQL COMMUNICATION AREA               * 00880000
*                                                                    * 00890000
*   TABLES = NONE                                                    * 00900000
*                                                                    * 00910000
*   CHANGE-ACTIVITY = NONE                                           * 00920000
*                                                                    * 00930000
*                                                                    * 00940000
*  *PSEUDOCODE*                                                      * 00950000
*                                                                    * 00960000
*      THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR       * 00970000
*      DETAIL MODULE(S) ARE TO BE CALLED FOR THE IMS/PL1 ENVIRONMENT.* 00980000
*                                                                    * 00990000
*      WHAT HAS HAPPENED SO FAR?............. THE SUBSYSTEM          * 01000000
*      DEPENDENT MODULE (IMS,CICS) (SQL 0) HAS READ THE              * 01010000
*      INPUT SCREEN, FORMATTED THE INPUT, AND PASSED CONTROL         * 01020000
*      TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT   * 01030000
*      FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF    * 01040000
*      ALL SYSTEM FIELDS ARE VALID, SQL 1 PASSED CONTROL TO THIS     * 01050000
*      MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH     * 01060000
*      POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE    * 01070000
*      BETWEEN SQL 0 , SQL 1, SQL 2, AND THE SECONDARY SELECTION     * 01080000
*      AND DETAIL MODULES.                                           * 01090000
*                                                                    * 01100000
*      WHAT IS INCLUDED IN THIS MODULE?............                  * 01110000
*      ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.    * 01120000
*      ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE        * 01130000
*      SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND                * 01140000
*      SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. ALL CURSOR HOST    * 01150000
*      VARIABLES ARE DECLARED IN THIS PROCEDURE BECAUSE OF THE       * 01160000
*      RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE* 01170000
*      THE CURSOR DEFINITION.                                        * 01180000
*                                                                    * 01190000
*      PROCEDURE                                                     * 01200000
*         IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR              * 01210000
*           IS NOT WILLING TO ACCEPT AN ANSWER THEN                  * 01220000
*                          NEW REQUEST*                              * 01230000
*                                                                    * 01240000
*          ELSE                                                      * 01250000
*                 IF ANSWER TO A SECONDARY SELECTION THEN            * 01260000
*                          DETERMINE IF NEW REQUEST.                 * 01270000
*                                                                    * 01280000
*          CASE (NEW REQUEST)                                        * 01290000
```

```
 *                 SUBCASE ('ACTION')                                 *
 *                         DETAIL PROCESSOR                           *
 *                         RETURN TO SQL 1                            *
 *                 ENDSUB                                             *
 *                                                                    *
 *                 SUBCASE ('DISPLAY','ERASE','UPDATE')               *
 *                         CALL SECONDARY SELECTION                   *
 *                         IF # OF POSSIBLE CHOICES IS ^= 1 THEN      *
 *                                RETURN TO SQL 1                     *
 *                         ELSE                                       *
 *                                CALL THE DETAIL PROCESSOR           *
 *                                RETURN TO SQL 1.                    *
 *                 ENDSUB                                             *
 *                                                                    *
 *          ENDCASE                                                   *
 *                                                                    *
 *          IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS      *
 *          ACTUALLY BEEN MADE THEN                                   *
 *                      VALID SELECTION #?                            *
 *                      IF IT IS VALID THEN                           *
 *                            CALL DETAIL PROCESSOR                   *
 *                            RETURN TO SQL 1                         *
 *                      ELSE                                          *
 *                            PRINT ERROR MSG                         *
 *                            RETURN TO SQL 1.                        *
 *                                                                    *
 *          IF ANSWER TO SECONDARY SELECTION THEN                     *
 *               CALL SECONDARY SELECTION                             *
 *               RETURN TO SQL 1.                                     *
 *                                                                    *
 *      IF ANSWER TO DETAIL THEN                                      *
 *               CALL DETAIL PROCESSOR                                *
 *               RETURN TO SQL 1.                                     *
 *                                                                    *
 *    END.                                                            *
 *                                                                    *
 *     *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES   *
 *      THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.           *
 *------------------------------------------------------------------*/

  DCL DSN8MPG EXTERNAL ENTRY;
  DCL LENGTH BUILTIN;

   /* INCLUDE DECLARES */
   EXEC SQL INCLUDE DSN8MPCA;   /*COMMUNICATION AREA BETWEEN MODULES */
   EXEC SQL INCLUDE SQLCA;      /*SQL COMMUNICATION AREA             */
                               /* ORGANIZATION                       */
   EXEC SQL INCLUDE DSN8MPDP;   /* DCLGEN FOR DEPARTMENT             */
   EXEC SQL INCLUDE DSN8MPEM;   /* DCLGEN FOR EMPLOYEE               */
   EXEC SQL INCLUDE DSN8MPED;   /* DCLGEN FOR EMPLOYEE-DEPARTMENT    */
   EXEC SQL INCLUDE DSN8MPDM;   /* DCLGEN FOR DEPARTMENT/MANAGER     */
   EXEC SQL INCLUDE DSN8MPAD;   /* DCLGEN FOR ADMINISTRATION DETAIL  */
   EXEC SQL INCLUDE DSN8MPA2;   /* DCLGEN FOR ADMINISTRATION DETAIL  */
                               /* PROGRAMMING TABLES                 */
   EXEC SQL INCLUDE DSN8MPOV;   /* DCLGEN FOR OPTION VALIDATION      */
   EXEC SQL INCLUDE DSN8MPDT;   /* DCLGEN FOR DISPLAY TEXT TABLE     */

    /* CURSORS */
   EXEC SQL INCLUDE DSN8MPAL;    /* MAJSYS O - SEC SEL FOR DS AND DE */
   EXEC SQL INCLUDE DSN8MPAE;    /* MAJSYS O - SEC SEL FOR EM        */
   EXEC SQL INCLUDE DSN8MPDA;    /* MAJSYS O - DETAIL  FOR DS        */
   EXEC SQL INCLUDE DSN8MPDH;    /* PROG TABLES - DISPLAY HEADINGS   */

     /*******************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
     /*******************************************************/

 DCL  MODULE            CHAR (07) INIT ('DSN8IP2');
 DCL  OUTMSG            CHAR (69);

  /*******************************************************/
  /* SQL RETURN CODE HANDLING                            */
  /*******************************************************/

   EXEC SQL WHENEVER SQLERROR   GO TO DB_ERROR;
   EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

0  DCL UNSPEC BUILTIN;
   DCL VERIFY BUILTIN;

  /*******************************************************/
  /*   INITIALIZATIONS                                   */
```

```
/*********************************************************/

 DSN8_MODULE_NAME.MAJOR='DSN8IP2';
 DSN8_MODULE_NAME.MINOR=' ';

   /*********************************************************/
   /*  DETERMINES WHETHER NEW REQUEST OR NOT               */
   /*********************************************************/

/* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */
/* NEW REQUEST.                                                 */

 IF PCONVSTA.PREV = ' ' THEN
   COMPARM.NEWREQ = 'Y';

/* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A  */
/* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/
/* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */

 IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' &
    SUBSTR(COMPARM.DATA,3,58) ^= ' ' &
    COMPARM.DATA ^= 'NEXT'
    THEN COMPARM.NEWREQ = 'Y';

   /*********************************************************/
   /*  IF NEW REQUEST AND ACTION IS 'ADD' THEN              */
   /*       CALL DETAIL PROCESSOR                           */
   /*  ELSE CALL SECONDARY SELECTION                        */
   /*********************************************************/

 IF COMPARM.NEWREQ='Y' THEN
   DO;
     IF COMPARM.ACTION = 'A' THEN
       DO;
         CALL DETAIL;                /* CALL DETAIL PROCESSOR    */
         GO TO EXIT;                 /* RETURN                   */
       END;

     CALL SECSEL;                    /* CALL SECONDARY SELECTION */

     IF MAXSEL = 1 THEN              /* IF NO. OF CHOICES = 1    */
       CALL DETAIL;                  /* CALL DETAIL PROCESSOR    */
     GO TO EXIT;                     /* RETURN                   */
   END;

/* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */
/* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS                */
/* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2     */
/* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION.            */

   /*********************************************************/
   /*  DETERMINES IF VALID SELECTION NUMBER                 */
   /*********************************************************/

 IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */

 IF PCONVSTA.MAXSEL < 1  THEN GO TO IP201; /* NO VALID CHOICES */

 IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROOL REQUEST*/

 IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2)
                THEN GO TO IP201;  /* NO CHANGE ON INPUT SCREEN */

 IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN     /* SECOND CHAR BLANK */
    IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN DO;
        SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1);
        SUBSTR(COMPARM.DATA,1,1) = '0';
        END;

 IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 &
           SUBSTR(COMPARM.DATA,1,2) > '00' THEN
       IF DATAP <= PCONVSTA.MAXSEL THEN DO;
          COMPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */
          CALL DETAIL;          /* CALL DETAIL PROCESSOR         */
          GO TO EXIT;           /* RETURN                        */
          END;

                              /* INVALID SELECTION NO.       */
                              /* PRINT ERROR MESSAGE         */
        CALL DSN8MPG (MODULE, '072E', OUTMSG);
        PCONVSTA.MSG = OUTMSG;
```

```
   GO TO EXIT;                      /* RETURN                     */

     /**********************************************************/
     /*  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL       */
     /**********************************************************/

  /* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */
  IP201:
   IF PCONVSTA.PREV = 'S' THEN
     DO;
       CALL SECSEL;                 /* CALL SECONDARY SELECTION   */
       GO TO EXIT;                  /* RETURN                     */
     END;
   IF PCONVSTA.PREV = 'D' THEN
     DO;
       CALL DETAIL;                 /* CALL DETAIL PROCESSOR      */
       GO TO EXIT;                  /* RETURN                     */
     END;

                                            /* LOGIC ERROR        */
   CALL DSN8MPG (MODULE, '066E', OUTMSG);  /* PRINT ERROR MESSAGE*/
   PCONVSTA.MSG = OUTMSG;
   GO TO EXIT;

    EXEC SQL INCLUDE DSN8MPXX;      /* HANDLES SQL ERRORS         */
    GO TO EXIT;                     /* RETURN                     */

     /***********************************************************/
     /*  CALLS SECONDARY SELECTION AND RETURNS TO SQL 1          */
     /* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
     /***********************************************************/

 SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE */
       PCONVSTA.LASTSCR = 'DSN8001';      /* NOTE GENERAL SCREEN */
       IF COMPARM.OBJFLD='DS' |           /* DEPARTMENT STRUCTURE*/

          COMPARM.OBJFLD='DE' |           /* INDIVIDUAL DEPARTMENT*/

          COMPARM.OBJFLD='EM' THEN        /* INDIVIDUAL EMPLOYEE  */
         DO;
           CALL DSN8MPA;
           RETURN;
         END;
                                              /*MISSING SECONDARY SEL*/
   CALL DSN8MPG (MODULE, '062E', OUTMSG);  /* PRINT ERROR MESSAGE*/
   PCONVSTA.MSG = OUTMSG;
   GO TO EXIT;
   END SECSEL;

      /**********************************************************/
      /* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1            */
      /**********************************************************/

 DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */
       PCONVSTA.LASTSCR = 'DSN8002';      /* NOTE DETAIL  SCREEN   */

       IF COMPARM.OBJFLD='DS' THEN        /* ADMINISTRATIVE        */
         DO;                              /* DEPARTMENT STRUCTURE  */
           CALL DSN8MPD;
           RETURN;
         END;

       IF COMPARM.OBJFLD='DE' THEN        /* INDIVIDUAL DEPARTMENT */
         DO;                              /* PROCESSING            */
           CALL DSN8MPE;
           RETURN;
         END;

       IF COMPARM.OBJFLD='EM' THEN        /* INDIVIDUAL EMPLOYEE    */
         DO;                              /* PROCESSING             */
           CALL DSN8MPF;
           RETURN;
         END;

                                          /*MISSING DETAIL MODULE*/
   CALL DSN8MPG (MODULE, '062E', OUTMSG);  /* PRINT ERROR MESSAGE*/
   PCONVSTA.MSG = OUTMSG;
   GO TO EXIT;                            /* RETURN             */
   END DETAIL;

  EXIT: RETURN;
                            /* ORGANIZATION */
```

```
       EXEC SQL INCLUDE DSN8MPA;       /* SEC SEL - ADMIN STRUCTURE */ 03750000
       EXEC SQL INCLUDE DSN8MPD;       /* DETAIL  - ADMIN STRUCTURE */ 03760000
       EXEC SQL INCLUDE DSN8MPE;       /* DETAIL  - DEPARTMENTS */     03770000
       EXEC SQL INCLUDE DSN8MPF;       /* DETAIL  - EMPLOYEES   */     03780000
  END;                                 /*       DSN8IP2         */     03790000
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSN8IP6

THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT, CALLS DSN8IP7, FORMATS OUTPUT MESSAGE AND SENDS IT.

```
DSN8IP6: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN);                     00010000
/******************************************************************** 00020000
*                                                                     * 00030000
*   MODULE NAME = DSN8IP6                                             * 00040000
*                                                                     * 00050000
*   DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                         * 00060000
*                      SUBSYSTEM INTERFACE MODULE                     * 00070000
*                      IMS                                            * 00080000
*                      PL/I                                           * 00090000
*                      PROJECT                                        * 00100000
*                                                                     * 00110000
*    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985           * 00120000
*    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083            * 00130000
*                                                                     * 00140000
*    STATUS = RELEASE 2, LEVEL 0                                      * 00150000
*                                                                     * 00160000
*   FUNCTION = THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT,   * 00170000
*              CALLS DSN8IP7, FORMATS OUTPUT MESSAGE AND SENDS IT.    * 00180000
*                                                                     * 00190000
*   NOTES =  NONE                                                     * 00200000
*                                                                     * 00210000
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                             * 00220000
*      PROCESSOR   = PL/I OPTIMIZER                                   * 00230000
*      MODULE SIZE = SEE LINKEDIT                                     * 00240000
*      ATTRIBUTES  = REUSABLE                                         * 00250000
*                                                                     * 00260000
*   ENTRY POINT =  DSN8IP6                                            * 00270000
*      PURPOSE = SEE FUNCTION                                         * 00280000
*      LINKAGE = FROM IMS                                             * 00290000
*                                                                     * 00300000
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:         * 00310000
*         COMMON AREA:                                                * 00320000
*                                                                     * 00330000
*               SYMBOLIC LABEL/NAME =  COMPARM.PFKIN                  * 00340000
*               DESCRIPTION = 00/01/02/03/08/10                       * 00350000
*                                                                     * 00360000
*               SYMBOLIC LABEL/NAME =  COMPARM.INAREA                 * 00370000
*               DESCRIPTION = USER INPUT                              * 00380000
*                                                                     * 00390000
*         INPUT-MESSAGE:                                              * 00400000
*                                                                     * 00410000
*               SYMBOLIC LABEL/NAME  =  DSN8IPFI                      * 00420000
*               DESCRIPTION = GENERAL MENU                            * 00430000
*                                                                     * 00440000
*               SYMBOLIC LABEL/NAME  =  DSN8IPEI                      * 00450000
*               DESCRIPTION = SECONDARY SELECTION MENU                * 00460000
*                                                                     * 00470000
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                       * 00480000
*         COMMON AREA:                                                * 00490000
*                                                                     * 00500000
*               SYMBOLIC LABEL/NAME =  COMPARM.OUTAREA                * 00510000
*               DESCRIPTION = USER OUTPUT                             * 00520000
*                                                                     * 00530000
*               SYMBOLIC LABEL/NAME = COMPARM.LASTSCR                 * 00540000
*               DESCRIPTION = DSN8001/DSN8002                         * 00550000
*                                                                     * 00560000
*         OUTPUT-MESSAGE:                                             * 00570000
*                                                                     * 00580000
*               SYMBOLIC LABEL/NAME =  DSN8IPFO                       * 00590000
*               DESCRIPTION = GENERAL MENU                            * 00600000
*                                                                     * 00610000
*               SYMBOLIC LABEL/NAME =  DSN8IPEO                       * 00620000
*               DESCRIPTION = SECONDARY SELECTION MENU                * 00630000
```

```
 *                                                                   *
 *   EXIT-NORMAL =                                                   *
 *                                                                   *
 *   EXIT-ERROR =                                                    *
 *                                                                   *
 *      RETURN CODE = NONE                                           *
 *                                                                   *
 *      ABEND CODES =  NONE                                          *
 *                                                                   *
 *      ERROR-MESSAGES =                                             *
 *         DSN8064E - INVALID DL/I STC-CODE ON GU MSG                *
 *         DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG              *
 *                                                                   *
 *   EXTERNAL REFERENCES =                                           *
 *      ROUTINES/SERVICES =  MODULE DSN8IP7                          *
 *                           MODULE PLITDLI                          *
 *                           MODULE DSN8MPG                          *
 *                                                                   *
 *      DATA-AREAS =                                                 *
 *         DSN8MPCA            - PARAMETER TO BE PASSED TO DSN8CP7   *
 *                               CONTAINS TERMINAL INPUT AND         *
 *                               OUTPUT AREAS.                       *
 *         IN_MESSAGE          - MFS INPUT                           *
 *         OUT_MESSAGE         - MFS OUTPUT                          *
 *                                                                   *
 *      CONTROL-BLOCKS =  NONE                                       *
 *                                                                   *
 *   TABLES =  NONE                                                  *
 *                                                                   *
 *   CHANGE-ACTIVITY =  NONE                                         *
 *                                                                   *
 *  *PSEUDOCODE*                                                     *
 *     PROCEDURE                                                     *
 *           DECLARATIONS.                                           *
 *             ALLOCATE PL/I WORK AREA FOR COMMAREA.                 *
 *             INITIALIZATION.                                       *
 *             PUT MODULE NAME 'DSN8IP6' IN AREA USED BY ERROR-HANDLER*
 *             PUT MODNAME 'DSN8IPFO' IN MODNAME FIELD.              *
 *                                                                   *
 *           STEP1.                                                  *
 *           CALL DLI GU INPUT MESSAGE.                              *
 *           IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND       *
 *              STOP PROGRAM.                                        *
 *                                                                   *
 *           IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.     *
 *              MOVE INPUT MESSAGE FIELDS TO CORRESPONDING           *
 *                   INAREA FIELDS IN COMPARM.                       *
 *              CALL DSN8IP7 (COMMAREA)                              *
 *              MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING     *
 *                   OUTPUT MESSAGE FIELDS.                          *
 *              IF LASTSCR 'DSN8001' MOVE 'DSN8IPFO' TO MODNAME FIELD *
 *                   ELSE MOVE 'DSN8IPEO' TO MODNAME FIELD.          *
 *                                                                   *
 *           CALL DLI ISRT OUTPUT MESSAGE.                           *
 *           IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND       *
 *              STOP PROGRAM.                                        *
 *                                                                   *
 *     END.                                                          *
 *                                                                   *
 *-------------------------------------------------------------------*
     /*******************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
     /*******************************************************/

 DCL  MODULE            CHAR (07) INIT ('DSN8IP6');
 DCL  OUTMSG            CHAR (69);

1/*******************************************************************/
 /*       DECLARATION FOR INPUT:  MIDNAME DSN8IPFI/DSN8IPEI         */
 /*******************************************************************/
0DCL  1 IN_MESSAGE     STATIC,
        2 LL             BIN FIXED (31),
        2 Z1             CHAR (1),
        2 Z2             CHAR (1),
        2 TC_CODE        CHAR (7),
        2 MESSAGE,
          3 INPUT,
            5 MAJSYS       CHAR (1),
            5 ACTION       CHAR (1),
            5 OBJFLD       CHAR (2),
            5 SEARCH       CHAR (2),
            5 PFKIN        CHAR (2),
```

```
            5 DATA        CHAR (60),
            5 TRANDATA(15) CHAR (40);
-/*********************************************************************/
 /*        DECLARATION FOR OUTPUT: MODNAME DSN8IPFO/DSN8IPEO          */
 /*********************************************************************/
0DCL  1 OUT_MESSAGE    STATIC,
        2 LL           BIN FIXED (31) INIT (1613),
        2 ZZ           BIN FIXED (15) INIT (0),
        2 OUTPUT,
          3 OUTPUTAREA,
            5 MAJSYS       CHAR (1),
            5 ACTION       CHAR (1),
            5 OBJFLD       CHAR (2),
            5 SEARCH       CHAR (2),
            5 DATA         CHAR (60),
            5 TITLE        CHAR (50),
            5 DESC2        CHAR (50),
            5 DESC3        CHAR (50),
            5 DESC4        CHAR (50),
            5 MSG          CHAR (79),
            5 PFKTEXT      CHAR (79),
            5 OUTPUT,
              7 LINE (15)  CHAR (79);
1/*********************************************************************/
 /*        DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE      */
 /*        SUBSYSTEM DEPENDENT MODULE IMS/DL1 AND SQL1 AND SQL2       */
 /*********************************************************************/
      EXEC SQL INCLUDE DSN8MPCA;
1/*********************************************************************/
 /*        DECLARATION FOR PGM-LOGIC                                  */
 /*********************************************************************/
0DCL  ONE        BIN FIXED (31) INIT (1)  STATIC;
 DCL  THREE      BIN FIXED (31) INIT (3)  STATIC;
 DCL  FOUR       BIN FIXED (31) INIT (4)  STATIC;
0DCL  GU_FKT     CHAR (4) INIT ('GU  ') STATIC;
 DCL  ISRT_FKT   CHAR (4) INIT ('ISRT') STATIC;
 DCL  CHNG_FKT   CHAR (4) INIT ('CHNG') STATIC;
 DCL  ROLL_FKT   CHAR (4) INIT ('ROLL') STATIC;
0DCL  MODNAME    CHAR (8) STATIC;
0DCL  (ADDR,LOW) BUILTIN;
0DCL  PLITDLI EXTERNAL ENTRY;
 DCL  DSN8IP7 EXTERNAL ENTRY;
0DCL  (IOPCB_ADDR,ALTPCB_ADDR) POINTER;
0DCL  DSN8MPG EXTERNAL ENTRY;
1/*********************************************************************/
 /*        DECLARATION FOR IO / ALTPCB MASK                           */
 /*********************************************************************/
0DCL  1 IOPCB        BASED (IOPCB_ADDR),
        2 IOLTERM    CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2),
        2 CDATE      CHAR (4),
        2 CTIME      CHAR (4),
        2 SEQNUM     CHAR (4),
        2 MOD_NAME   CHAR (8),
        2 USERID     CHAR (8);
0DCL  1 ALTPCB       BASED (ALTPCB_ADDR),
        2 ALTLTERM   CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2);

     /***************************************************************/
     /*         ALLOCATE COBOL WORK AREA /INITIALIZATIONS           */
     /***************************************************************/

0   ALLOCATE COMMAREA SET(COMMPTR);
         COMMAREA   = '';                       /* CLEAR COMMON AREA*/
         IN_MESSAGE = '';                       /* CLEAR INPUT FIELD*/
         MODNAME                = 'DSN8IPFO';  /* GET MODULE  NAME */
         DSN8_MODULE_NAME.MAJOR = 'DSN8IP6';   /* GET MODULE NAME  */
         OUTAREA.MAJSYS = 'P';                 /* MAJOR SYSTEM - P */

     /***************************************************************/
     /*     CALL DL1 GU INPUT MESSAGE                               */
     /*     PRINT ERROR MESSAGE IF STATUS CODE NOT OK               */
     /***************************************************************/

0   CALL PLITDLI (THREE,GU_FKT,IOPCB,IN_MESSAGE);   /* CALL DL1 GU   */

0   IF   IOPCB.STC_CODE ^= ' ' THEN                 /* ERROR?        */
         DO;
                                                    /* PRINT MESSAGE */
```

```
          CALL DSN8MPG (MODULE, '064E', OUTMSG);
          OUTPUTAREA.MSG = OUTMSG||
                           IOPCB.STC_CODE;

          GO TO CSEND;                /*CALL DL1 ISRT OUTPUT MESSAGE */
         END;

     /***************************************************************/
     /*     CLEARED AND UNFORMATTED SCREEN?                         */
     /***************************************************************/

    IF   Z2 = LOW(1) THEN COMPARM.PFKIN   = '00';
0        PCONVSTA.CONVID  = IOPCB.IOLTERM||USERID;
         INAREA           = INPUT, BY NAME;     /*MOVE INPUT MESSAGE */
         INAREA.MAJSYS    = 'P';            /*FIELDS TO INAREA FIELDS*/

0   CALL DSN8IP7 (COMMPTR);

                                                 /*MOVE OUTAREA FIELDS */
0        OUTPUTAREA = OUTAREA , BY NAME;   /*TO OUTPUT MESSAGE FIELDS*/

0   IF   LASTSCR = 'DSN8002' THEN MODNAME = 'DSN8IPEO';
                               ELSE MODNAME = 'DSN8IPFO';

     /***************************************************************/
     /*   CALL DL ISRT OUTPUT MESSAGE                               */
     /*   PRINT ERROR MESSAGE IF STATUS CODE NOT OK                 */
     /***************************************************************/

 CSEND:

                                                       /* CALL DL1 ISRT*/
    CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME);


0   IF   IOPCB.STC_CODE = ' ' THEN GO TO CEND;      /* STATUS CODE OK*/

                                                  /*PRINT ERROR MESSAGE*/
    CALL DSN8MPG (MODULE, '065E', OUTMSG);
    OUTPUTAREA.MSG = OUTMSG||IOPCB.STC_CODE;

0   CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM);   /* CALL DL1 CHNG */

0   IF   ALTPCB.STC_CODE ^= ' ' THEN GO TO CSEND1;  /* ERROR?        */

                                                 /* CALL DL1 ISRT    */
0   CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);

0CSEND1:                                         /* PERFORM ROLLBACK */
    CALL PLITDLI (ONE,ROLL_FKT);

0CEND:                                           /* RETURN           */
    END DSN8IP6;
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSN8IP7

THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS THE PARAMETER AREA.

```
 DSN8IP7:PROC (COMMPTR) ;
 /*******************************************************************
 *                                                                 *
 *   MODULE NAME = DSN8IP7                                         *
 *                                                                 *
 *   DESCRIPTIVE NAME = SAMPLE APPLICATION                         *
 *                      SQL 1 MAINLINE                             *
 *                      IMS                                        *
 *                      PL/I                                       *
 *                                                                 *
 *    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985       *
 *    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083        *
 *                                                                 *
 *    STATUS = RELEASE 2, LEVEL 0                                  *
 *                                                                 *
```

```
*   FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE     *
*              SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS       *
*              THE PARAMETER AREA.                                   *
*              INCLUDE DSN8MP1.                                      *
*              CALL DSN8IP8.                                         *
*              RETURN TO DSN8IP6.                                    *
*                                                                    *
*   NOTES =  NONE                                                    *
*                                                                    *
*   MODULE TYPE = PL/I PROC(COMMPTR).                                *
*      PROCESSOR = DB2  PRECOMPILER,  PL/I OPTIMIZER                 *
*      MODULE SIZE = SEE LINKEDIT                                    *
*      ATTRIBUTES = REUSABLE                                         *
*                                                                    *
*   ENTRY POINT = DSN8IP7                                            *
*      PURPOSE = SEE FUNCTION                                        *
*      LINKAGE = CALLED BY DSN8IP6                                   *
*                                                                    *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:        *
*                                                                    *
*                SYMBOLIC LABEL/NAME = COMMPTR                       *
*                DESCRIPTION = POINTER TO COMMAREA                   *
*                                                                    *
*              COMMON AREA.                                          *
*                                                                    *
*                SYMBOLIC LABEL/NAME = PFKIN                         *
*                DESCRIPTION = 00/01/02/03/07/08/10                  *
*                                                                    *
*                SYMBOLIC LABEL/NAME = INAREA                        *
*                DESCRIPTION = USER INPUT                            *
*                                                                    *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                      *
*               COMMON AREA.                                         *
*                                                                    *
*                SYMBOLIC LABEL/NAME = OUTAREA                       *
*                DESCRIPTION =GENERAL MENU OR                        *
*                             SECONDARY SELECTION MENU               *
*                                                                    *
*                SYMBOLIC LABEL/NAME = LASTSCR                       *
*                DESCRIPTION = DSN8001/DSN8002                       *
*                                                                    *
*                                                                    *
*   EXIT-NORMAL = DSN8IP6                                            *
*                                                                    *
*   EXIT-ERROR = DSN8IP6                                             *
*                                                                    *
*      RETURN CODE = NONE                                            *
*                                                                    *
*      ABEND CODES =  NONE                                           *
*                                                                    *
*      ERROR-MESSAGES =  NONE                                        *
*                                                                    *
*   EXTERNAL REFERENCES =                                            *
*      ROUTINES/SERVICES =    NONE                                   *
*                                                                    *
*      DATA-AREAS =                                                  *
*         DSN8MPCA            - PLI STRUCTURE FOR COMMAREA           *
*         DSN8MPCS            - VCONA TABLE DCL AND PCONA DCLGEN     *
*         DSN8MPOV            - VOPTVAL TABLE DCL & POPTVAL DCLGEN   *
*         DSN8MPVO            - VALIDATION CURSORS                   *
*         DSN8MP1             - SQL1 COMMON MODULE FOR IMS AND CICS  *
*         DSN8MP3 -- DSN8MP5  - VALIDATION MODULES CALLED BY DSN8MP1 *
*         DSN8MPXX            - SQL ERROR HANDLER                    *
*                                                                    *
*      CONTROL-BLOCKS =                                              *
*         SQLCA               - SQL COMMUNICATION AREA               *
*                                                                    *
*   TABLES =  NONE                                                   *
*                                                                    *
*   CHANGE-ACTIVITY =  NONE                                          *
*                                                                    *
*                                                                    *
*  *PSEUDOCODE*                                                      *
*    PROCEDURE                                                       *
*      INCLUDE DECLARATIONS.                                         *
*      INCLUDE DSN8MP1.                                              *
*      INCLUDE ERROR HANDLER.                                        *
*                                                                    *
*      CP1EXIT:  ( REFERENCED BY DSN8MP1 )                           *
*          RETURN.                                                   *
*                                                                    *
*      CP1CALL:  ( REFERENCED BY DSN8MP1 )                           *
```

```
 *          CALL 'DSN8IP8'(COMMPTR).                                  *
 *          GO TO MP1SAVE.  (LABEL IN DSN8MP1)                        *
 *                                                                    *
 *      INCLUDE VALIDATION MODULES.                                   *
 *                                                                    *
 *     END.                                                           *
 *********************************************************************/
1/*********************************************************************/
 /*                        SQL1   MAINLINE                           */
 /*********************************************************************/
                                          /* SQL RETURN CODE HANDLING*/
 EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
 EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

 /*********************************************************/
 /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
 /*********************************************************/

 DCL  MODULE            CHAR (07) INIT ('DSN8IP7');
 DCL  OUTMSG            CHAR (69);


 DCL STRING BUILTIN;
 DCL J FIXED BIN;
 DCL SAVE_CONVID CHAR(16);
 DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);
 DCL DSN8IP8 EXTERNAL ENTRY;
 DCL DSN8MPG EXTERNAL ENTRY;
 EXEC SQL INCLUDE DSN8MPCA;               /* INCLUDE COMMAREA */
 DSN8_MODULE_NAME.MAJOR = 'DSN8IP7 ';   /* INITIALIZE MODULE NAME*/
 EXEC SQL INCLUDE DSN8MPCS;               /* INCLUDE PCONA    */
 EXEC SQL INCLUDE DSN8MPOV;               /* INCLUDE POPTVAL  */
 EXEC SQL INCLUDE DSN8MPVO;               /* INCLUDE CURSOR   */
 EXEC SQL INCLUDE SQLCA;                  /* INCLUDE SQL COMMAREA*/
 EXEC SQL INCLUDE DSN8MP1;                /* INCLUDE SQL1 MAIN*/
 EXEC SQL INCLUDE DSN8MPXX;               /* INCLUDE ERRORHANDLER */

 CP1EXIT :
      RETURN;                             /* EXIT */

 CP1CALL :
                                          /* GO TO DSN8IP8 (SQL2)     */
      CALL DSN8IP8 (COMMPTR);
      GO TO MP1SAVE;

 EXEC SQL INCLUDE DSN8MP3;                /* INCLUDE ACTION VALIDATION*/
 EXEC SQL INCLUDE DSN8MP4;                /* INCLUDE OBJECT VALIDATION*/
 EXEC SQL INCLUDE DSN8MP5;                /* INCLUDE SEARCH CRITERIA*/
 END;                                     /* VALIDATION             */
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSN8IP8

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN CALLS SECONDARY SELECTION MODULES DSN8MPM CALLS DETAIL MODULES DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ CALLED BY DSN8IP7 (SQL1) .

```
   DSN8IP8: PROC(COMMPTR) ;                                            00010000
                                                              %PAGE;   00020000
 /******************************************************************** 00030000
 *                                                                   * 00040000
 *   MODULE NAME = DSN8IP8                                           * 00050000
 *                                                                   * 00060000
 *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                      * 00070000
 *                      SQL 2 COMMON MODULE                          * 00080000
 *                      IMS                                          * 00090000
 *                      PL/I                                         * 00100000
 *                      PROJECT                                      * 00110000
 *                                                                   * 00120000
 *     LICENSED MATERIALS - PROPERTY OF IBM                          * 00130000
 *     5695-DB2                                                      * 00136000
 *     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.      * 00143000
 *                                                                   * 00150000
 *     STATUS = VERSION 4                                            * 00160000
```

```
*                                                                        * 00170000
*   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN* 00180000
*              CALLS SECONDARY SELECTION MODULES                         * 00190000
*                    DSN8MPM                                             * 00200000
*              CALLS DETAIL MODULES                                      * 00210000
*                    DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ             * 00220000
*              CALLED BY DSN8IP7 (SQL1)                                  * 00230000
*                                                                        * 00240000
*   NOTES =    NONE                                                      * 00250000
*                                                                        * 00260000
*   MODULE TYPE = BLOCK OF PL/I CODE                                     * 00270000
*      PROCESSOR = DB2  PRECOMPILER, PL/I OPTIMIZER                      * 00280000
*      MODULE SIZE = SEE LINKEDIT                                        * 00290000
*      ATTRIBUTES = REUSABLE                                             * 00300000
*                                                                        * 00310000
*   ENTRY POINT = DSN8IP8                                                * 00320000
*      PURPOSE = SEE FUNCTION                                            * 00330000
*      LINKAGE = NONE                                                    * 00340000
*       INPUT =                                                          * 00350000
*               SYMBOLIC LABEL/NAME = COMMPTR                            * 00360000
*               DESCRIPTION = POINTER TO COMMAREA                        * 00370000
*                                                                        * 00380000
*      OUTPUT =                                                          * 00390000
*               SYMBOLIC LABEL/NAME = COMMPTR                            * 00400000
*               DESCRIPTION = POINTER TO COMMAREA                        * 00410000
*                                                                        * 00420000
*   EXIT-NORMAL =                                                        * 00430000
*                                                                        * 00440000
*   EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION       * 00450000
*                SPECIFIED IN DSN8IP8 WILL BE RAISED AND PROGRAM         * 00460000
*                WILL GO TO THE LABEL DB_ERROR.                          * 00470000
*                                                                        * 00480000
*                                                                        * 00490000
*      RETURN CODE = NONE                                                * 00500000
*                                                                        * 00510000
*      ABEND CODES = NONE                                                * 00520000
*                                                                        * 00530000
*      ERROR-MESSAGES =                                                  * 00540000
*              DSN8062E-AN OBJECT WAS NOT SELECTED                       * 00550000
*              DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR                   * 00570000
*              DSN8072E-INVALID SELECTION ON SECONDARY SCREEN            * 00580000
*                                                                        * 00590000
*   EXTERNAL REFERENCES = NONE                                           * 00600000
*      ROUTINES/SERVICES = 8 MODULES LISTED ABOVE                        * 00610000
*                  DSN8MPG     -  ERROR MESSAGE ROUTINE                  * 00620000
*                                                                        * 00630000
*      DATA-AREAS =                                                      * 00640000
*                  DSN8MPAC    -  DCLGEN FOR ACTIVITY TYPES              * 00650000
*                  DSN8MPAS    -  CURSOR SECONDARY SELECTION FOR         * 00660000
*                                 STAFF                                  * 00670000
*                  DSN8MPCA    -  COMMUNICATION AREA BETWEEN MODULES * 00680000
*                  DSN8MPDH    -  CURSOR FOR DISPLAY TEXT FROM           * 00690000
*                                 TDSPTXT TABLE                          * 00700000
*                  DSN8MPDP    -  DCLGEN FOR DEPARTMENT                  * 00710000
*                  DSN8MPDT    -  DCLGEN FOR DISPLAY TEXT TABLE          * 00720000
*                  DSN8MPEM    -  DCLGEN FOR EMPLOYEE                    * 00730000
*                  DSN8MPEP    -  DCLGEN FOR PROJECT/STAFFING            * 00740000
*                  DSN8MPES    -  CURSOR SECONDARY SELECTION FOR         * 00750000
*                                 ESTIMATES                              * 00760000
*                  DSN8MPOV    -  DCLGEN FOR OPTION VALIDATION           * 00770000
*                  DSN8MPPA    -  DCLGEN FOR PROJECT/ACTIVITIES          * 00780000
*                  DSN8MPPD    -  DCLGEN FOR PROJ STRUCTURE DETAIL       * 00790000
*                  DSN8MPP2    -  DCLGEN FOR PROJ STRUCTURE DETAIL       * 00795000
*                  DSN8MPPE    -  CURSOR PROJECT DETAIL                  * 00800000
*                  DSN8MPPJ    -  DCLGEN FOR PROJECTS                    * 00810000
*                  DSN8MPPL    -  CURSOR PROJECT LIST                    * 00820000
*                  DSN8MPPR    -  DCLGEN FOR PROJ/RESP EMPLOYEE          * 00830000
*                  DSN8MPSA    -  DCLGEN FOR PROJ ACTIVITY LISTING       * 00850000
*                  DSN8MPFP    -  DCLGEN FOR PROJECT-EMPLOYEE            * 00855000
*                  DSN8MPED    -  DCLGEN FOR EMPLOYEE-DEPT               * 00857000
*                  DSN8MPSL    -  CURSOR STAFFING LIST                   * 00860000
*                  DSN8MPS2    -  DCLGEN FOR PROJ ACTIVITY LISTING       * 00870000
*                  DSN8MPM     -  SECONDARY SELECTION FOR PROJECTS       * 00880000
*                  DSN8MPT     -  PROJECT ACTIVITY LIST                  * 00890000
*                  DSN8MPV     -  PROJECT STRUCTURE DETAIL               * 00900000
*                  DSN8MPW     -  ACTIVITY STAFFING DETAIL               * 00910000
*                  DSN8MPX     -  ACTIVITY ESTIMATE DETAIL               * 00920000
*                  DSN8MPZ     -  PROJECT DETAIL                         * 00930000
*                                                                        * 00940000
*      CONTROL-BLOCKS =                                                  * 00950000
*         SQLCA           - SQL COMMUNICATION AREA                       * 00960000
*                                                                        * 00970000
```

```
*   TABLES =  NONE
*                                                                    *
*   CHANGE-ACTIVITY =  NONE                                          *
*                                                                    *
*                                                                    *
*  *PSEUDOCODE*                                                      *
*                                                                    *
*      THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR       *
*      DETAIL MODULE(S) ARE TO BE CALLED IN THE IMS/PL/I             *
*      ENVIRONMENT.                                                  *
*                                                                    *
*      WHAT HAS HAPPENED SO FAR?.............THE SUBSYSTEM           *
*      DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS                *
*      READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL *
*      TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT   *
*      FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF    *
*      ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS      *
*      MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH     *
*      POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE    *
*      BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION      *
*      AND DETAIL MODULES.                                           *
*                                                                    *
*      WHAT IS 'INCLUDED' IN THIS MODULE?.............               *
*      ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.    *
*      ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE        *
*      SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND                *
*      SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE     *
*      RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE*
*      THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED  *
*      IN THIS PROCEDURE.                                            *
*                                                                    *
*    PROCEDURE                                                       *
*         IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR              *
*         IS NOT WILLING TO ACCEPT AN ANSWER THEN                    *
*                         NEW REQUEST*                               *
*                                                                    *
*         ELSE                                                       *
*                IF ANSWER TO A SECONDARY SELECTION THEN             *
*                         DETERMINE IF NEW REQUEST.                  *
*                                                                    *
*         CASE (NEW REQUEST)                                         *
*                                                                    *
*                SUBCASE ('ADD')                                     *
*                        DETAIL PROCESSOR                            *
*                        RETURN TO SQL 1                             *
*                ENDSUB                                              *
*                                                                    *
*                SUBCASE ('DISPLAY','ERASE','UPDATE')                *
*                        CALL SECONDARY SELECTION                    *
*                        IF # OF POSSIBLE CHOICES IS ^= 1 THEN       *
*                               RETURN TO SQL 1                      *
*                        ELSE                                        *
*                               CALL THE DETAIL PROCESSOR            *
*                               RETURN TO SQL 1                      *
*                ENDSUB                                              *
*                                                                    *
*         ENDCASE                                                    *
*                                                                    *
*         IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS       *
*         ACTUALLY BEEN MADE THEN                                    *
*                     VALID SELECTION #?                             *
*                     IF IT IS VALID THEN                            *
*                           CALL DETAIL PROCESSOR                    *
*                           RETURN TO SQL 1                          *
*                     ELSE                                           *
*                           PRINT ERROR MSG                          *
*                           RETURN TO SQL 1.                         *
*                                                                    *
*         IF ANSWER TO SECONDARY SELECTION THEN                      *
*                  CALL SECONDARY SELECTION                          *
*                  RETURN TO SQL 1.                                  *
*                                                                    *
*         IF ANSWER TO DETAIL THEN                                   *
*                  CALL DETAIL PROCESSOR                             *
*                  RETURN TO SQL 1.                                  *
*                                                                    *
*     END.                                                           *
*                                                                    *
*        *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES*
*         THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.        *
*                                                                    *
*    END.                                                            *
```

```
 *-------------------------------------------------------------------*/
   /* INCLUDE DECLARES */
   EXEC SQL INCLUDE DSN8MPCA;   /*COMMUNICATION AREA BETWEEN MODULES */
   EXEC SQL INCLUDE SQLCA;      /*SQL COMMUNICATION AREA             */
                               /* PROJECTS                           */
   EXEC SQL INCLUDE DSN8MPDP;   /* DCLGEN FOR DEPARTMENT             */
   EXEC SQL INCLUDE DSN8MPEM;   /* DCLGEN FOR EMPLOYEE               */
   EXEC SQL INCLUDE DSN8MPPJ;   /* DCLGEN FOR PROJECTS               */
   EXEC SQL INCLUDE DSN8MPAC;   /* DCLGEN FOR ACTIVITY TYPES         */
   EXEC SQL INCLUDE DSN8MPPA;   /* DCLGEN FOR PROJECT/ACTIVITIES     */
   EXEC SQL INCLUDE DSN8MPEP;   /* DCLGEN FOR PROJECT/STAFFING       */
   EXEC SQL INCLUDE DSN8MPPR;   /* DCLGEN FOR PROJ/RESP EMPLOYEE     */
   EXEC SQL INCLUDE DSN8MPPD;   /* DCLGEN FOR PROJ STRUCTURE DETAIL  */
   EXEC SQL INCLUDE DSN8MPP2;   /* DCLGEN FOR PROJ STRUCTURE DETAIL  */
   EXEC SQL INCLUDE DSN8MPSA;   /* DCLGEN FOR PROJ ACTIVITY LISTING  */
   EXEC SQL INCLUDE DSN8MPS2;   /* DCLGEN FOR PROJ ACTIVITY LISTING  */
   EXEC SQL INCLUDE DSN8MPFP;   /* DCLGEN FOR PROJECT-EMPLOYEE       */
   EXEC SQL INCLUDE DSN8MPED;   /* DCLGEN FOR EMPLOYEE-DEPT          */
                               /* PROGRAMMING TABLES                 */
   EXEC SQL INCLUDE DSN8MPOV;   /* DCLGEN FOR OPTION VALIDATION      */
   EXEC SQL INCLUDE DSN8MPDT;   /* DCLGEN FOR DISPLAY TEXT TABLE     */

    /* CURSORS */
   EXEC SQL INCLUDE DSN8MPPL;    /* MAJSYS P - SEC SEL FOR PS, AL, PR*/
   EXEC SQL INCLUDE DSN8MPES;    /* MAJSYS P - SEC SEL FOR AE        */
   EXEC SQL INCLUDE DSN8MPAS;    /* MAJSYS P - SEC SEL FOR AS        */
   EXEC SQL INCLUDE DSN8MPPE;    /* MAJSYS P - DETAIL  FOR PS        */
   EXEC SQL INCLUDE DSN8MPSL;    /* MAJSYS P - DETAIL  FOR AL        */
   EXEC SQL INCLUDE DSN8MPDH;    /* PROG TABLES - DISPLAY HEADINGS   */

   DCL LENGTH BUILTIN;
     /*********************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
     /*********************************************************/

   DCL  MODULE            CHAR (07) INIT ('DSN8IP8');
   DCL  OUTMSG            CHAR (69);

   DCL DSN8MPG EXTERNAL ENTRY;

  /*********************************************************/
  /* SQL RETURN CODE HANDLING                              */
  /*********************************************************/

   EXEC SQL WHENEVER SQLERROR   GO TO DB_ERROR;
   EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

0  DCL UNSPEC BUILTIN;
   DCL VERIFY BUILTIN;

  /*********************************************************/
  /*   INITIALIZATIONS                                     */
  /*********************************************************/

   DSN8_MODULE_NAME.MAJOR='DSN8IP8';
   DSN8_MODULE_NAME.MINOR=' ';

     /***********************************************************/
     /*  DETERMINES WHETHER NEW REQUEST OR NOT                  */
     /***********************************************************/

  /* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */
  /* NEW REQUEST.                                                 */

   IF PCONVSTA.PREV = ' ' THEN COMPARM.NEWREQ = 'Y';

  /* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A  */
  /* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/
  /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */

   IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' &
      SUBSTR(COMPARM.DATA,3,58) ^= ' ' &
      COMPARM.DATA ^= 'NEXT'
      THEN COMPARM.NEWREQ = 'Y';

     /***********************************************************/
     /*  IF NEW REQUEST AND ACTION IS 'ADD' THEN                */
     /*       CALL DETAIL PROCESSOR                             */
     /*  ELSE CALL SECONDARY SELECTION                          */
     /***********************************************************/

   IF COMPARM.NEWREQ='Y' THEN
```

```
    DO;
      IF COMPARM.ACTION = 'A' THEN
        DO;
          CALL DETAIL;                     /* CALL DETAIL PROCESSOR */
          GO TO EXIT;                      /* RETURN */
        END;

      CALL SECSEL;                 /* CALL SECONDARY SELECTION */

      IF MAXSEL = 1 THEN           /* IF NO. OF CHOICES = 1    */
        CALL DETAIL;               /* CALL DETAIL PROCESSOR    */
        GO TO EXIT;                /* RETURN                   */
    END;

 /* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */
 /* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS               */
 /* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2     */
 /* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION.            */

    /*********************************************************/
    /*  DETERMINES IF VALID SELECTION NUMBER                 */
    /*********************************************************/

  IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201;  /* TO SECONDARY SEL */

  IF PCONVSTA.MAXSEL < 1  THEN GO TO IP201;  /* NO VALID CHOICES */

  IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROLL REQUEST   */

  IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2)
               THEN GO TO IP201;  /* NO CHANGE ON INPUT SCREEN  */

  IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN     /* SECOND CHAR BLANK */

     IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN
      DO;
         SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1);
         SUBSTR(COMPARM.DATA,1,1) = '0';
      END;

  IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 &
            SUBSTR(COMPARM.DATA,1,2) > '00' THEN

        IF DATAP <= PCONVSTA.MAXSEL THEN
         DO;
            COMPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */
            CALL DETAIL;          /* CALL DETAIL PROCESSOR         */
            GO TO EXIT;           /* RETURN                        */
         END;

                                   /*   INVALID SECONDARY    */
                                   /*   SELECTION            */
                                   /*   PRINT ERROR MESSAGE  */
         CALL DSN8MPG (MODULE, '072E', OUTMSG);
         PCONVSTA.MSG = OUTMSG;
  GO TO EXIT;                       /*  RETURN               */

    /*********************************************************/
    /*  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL     */
    /*********************************************************/

 /* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */
 IP201:
  IF PCONVSTA.PREV = 'S' THEN
    DO;
      CALL SECSEL;                     /* CALL SECONDARY SELECTION*/
      GO TO EXIT;                      /* RETURN */
    END;

  IF PCONVSTA.PREV = 'D' THEN
    DO;
      CALL DETAIL;                    /* CALL DETAIL PROCESSOR */
      GO TO EXIT;                     /* RETURN */
    END;

                                      /*  LOGIC ERROR */
                                      /* PRINT ERROR MESSAGE */
  CALL DSN8MPG (MODULE, '066E', OUTMSG);
  PCONVSTA.MSG= OUTMSG;
  GO TO EXIT;                         /* RETURN */

   EXEC SQL INCLUDE DSN8MPXX;         /* HANDLES SQL ERRORS */
```

```
    GO TO EXIT;                        /* RETURN  */
     /************************************************************/
     /*  CALLS SECONDARY SELECTION AND RETURNS TO SQL 1          */
     /* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
     /************************************************************/
 SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE       */
       PCONVSTA.LASTSCR = 'DSN8001'; /* NOTE GENERAL SCREEN         */
       IF COMPARM.OBJFLD='AE' |      /*ACTIVITY ESTIMATE            */
          COMPARM.OBJFLD='AL' |      /*PROJECT ACTIVITY LISTING     */
          COMPARM.OBJFLD='AS' |      /*INDIVIDUAL PROJECT STAFFING  */
          COMPARM.OBJFLD='PR' |      /*INDIVIDUAL PROJECT PROCESSING*/
          COMPARM.OBJFLD='PS' THEN   /*PROJECT STRUCTURE            */
         DO;
           CALL DSN8MPM;           /*SECONDARY SELECTION FOR PROJECTS*/
           RETURN;                 /*RETURN */
         END;
                                      /* SECONDARY SELECTION MISSING */
                                        /* PRINT ERROR MESSAGE       */
     CALL DSN8MPG (MODULE, '062E', OUTMSG);
     PCONVSTA.MSG= OUTMSG;
     GO TO EXIT;
     END SECSEL;
      /***********************************************************/
      /* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1             */
      /***********************************************************/
 DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */
       PCONVSTA.LASTSCR = 'DSN8002';  /* SET FOR DETAIL  MAP       */
       IF COMPARM.OBJFLD='PS' THEN
         DO;
           CALL DSN8MPV;               /* PROJECT STRUCTURE DETAIL  */
           RETURN;
         END;
       IF COMPARM.OBJFLD='AL' THEN
         DO;
           CALL DSN8MPT;               /* PROJECT ACTIVITY LIST     */
           RETURN;
         END;
       IF COMPARM.OBJFLD='PR' THEN
         DO;
           CALL DSN8MPZ;                /* PROJECT DETAIL            */
           RETURN;
         END;
       IF COMPARM.OBJFLD='AE' THEN
         DO;
           CALL DSN8MPX;                /* ACTIVITY ESTIMATE DETAIL  */
           RETURN;
         END;
       IF COMPARM.OBJFLD='AS' THEN
         DO;
           CALL DSN8MPW;                /* ACTIVITY STAFFING DETAIL  */
           RETURN;
         END;
                                        /* MISSING DETAIL MODULE     */
                                        /* PRINT ERROR MESSAGE       */
   CALL DSN8MPG (MODULE, '062E', OUTMSG);
   PCONVSTA.MSG= OUTMSG;
   GO TO EXIT;
   END DETAIL;
  EXIT: RETURN;                         /* RETURNS TO SQL 1          */
                           /* PROJECTS */
        EXEC SQL INCLUDE DSN8MPM;       /* SEC SEL - PROJECTS        */
        EXEC SQL INCLUDE DSN8MPV;       /* DETAIL  - PROJ STRUCTURE  */
        EXEC SQL INCLUDE DSN8MPT;       /* DETAIL  - PROJ ACT LISTING*/
        EXEC SQL INCLUDE DSN8MPZ;       /* DETAIL  - INDIVIDUAL PROJ */
        EXEC SQL INCLUDE DSN8MPX;       /* DETAIL  - INDIVID ACTIVITY*/
        EXEC SQL INCLUDE DSN8MPW;       /* DETAIL  - INDIVID STAFFING*/
   END DSN8IP8;
```

Related reference

"Sample applications in IMS" on page 1332

A set of Db2 sample applications run in the IMS environment.

DSN8IP3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
DSN8IP3: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN);
/*******************************************************************
*                                                                  *
*   MODULE NAME = DSN8IP3                                          *
*                                                                  *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                     *
*                      PHONE APPLICATION                           *
*                      IMS                                         *
*                      PL/I                                        *
*                                                                  *
*     COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1991       *
*     SEE COPYRIGHT INSTRUCTIONS                                   *
*     LICENSED MATERIALS - PROPERTY OF IBM                         *
*                                                                  *
*     STATUS = VERSION 2 RELEASE 3, LEVEL 0                        *
*                                                                  *
*   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND        *
*              UPDATES THEM IF DESIRED.                            *
*                                                                  *
*   NOTES =                                                        *
*      DEPENDENCIES = TWO MFS MAPS ARE REQUIRED:                   *
*                     DSN8IPL AND DSN8IPN                          *
*      RESTRICTIONS = NONE                                         *
*                                                                  *
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                          *
*      PROCESSOR = DB2  PRECOMPILER, PL/I OPTIMIZER                *
*      MODULE SIZE = SEE LINKEDIT                                  *
*      ATTRIBUTES = REENTRANT                                      *
*                                                                  *
*   ENTRY POINT =  DSN8IP3                                         *
*      PURPOSE = SEE FUNCTION                                      *
*      LINKAGE = INVOKED FROM IMS                                  *
*                                                                  *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:      *
*              INPUT-MESSAGE:                                      *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = DSN8IPNO               *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)          *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = DSN8IPLO               *
*                     DESCRIPTION = PHONE MENU 2 (UPDATE)          *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = VPHONE                 *
*                     DESCRIPTION = VIEW OF TELEPHONE INFORMATION  *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = VEMPLP                 *
*                     DESCRIPTION = VIEW OF EMPLOYEE INFORMATION   *
*                                                                  *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                    *
*               OUTPUT-MESSAGE:                                    *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = DSN8IPNO               *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)          *
*                                                                  *
*                     SYMBOLIC LABEL/NAME = DSN8IPLO               *
*                     DESCRIPTION = PHONE MENU 2 (UPDATE)          *
*                                                                  *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION                  *
*                                                                  *
*   EXIT-ERROR =                                                   *
*                                                                  *
*      RETURN CODE = NONE                                          *
*                                                                  *
*      ABEND CODES =  NONE                                         *
*                                                                  *
*      ERROR-MESSAGES =                                            *
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED              *
*            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE   *
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE                 *
*            DSN8058I - PRESS PA1 FOR NEXT PAGE / ENTER FOR        *
*                       SELECTION MENU                             *
*            DSN8060E - SQL ERROR, RETURN CODE IS:                 *
*            DSN8064E - INVALID DL/I STC-CODE ON GU MSG            *
```

```
 *           DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG            *
 *                                                                   *
 *   EXTERNAL REFERENCES =                                           *
 *      ROUTINES/SERVICES =                                          *
 *         DSN8MPG             - ERROR MESSAGE ROUTINE               *
 *                                                                   *
 *      DATA-AREAS =                                                 *
 *         IN_MESSAGE          - MFS INPUT                           *
 *         OUT_MESSAGE         - MFS OUTPUT                          *
 *                                                                   *
 *      CONTROL-BLOCKS =                                             *
 *         SQLCA               - SQL COMMUNICATION AREA              *
 *                                                                   *
 *   TABLES = NONE                                                   *
 *                                                                   *
 *                                                                   *
 *   CHANGE-ACTIVITY = NONE                                          *
 *                                                                   *
 *                                                                   *
 *  *PSEUDOCODE*                                                     *
 *                                                                   *
 *  PROCEDURE                                                        *
 *      CALL DLI GU INPUT MESSAGE.                                   *
 *      IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE ,              *
 *         PGM-STOP.                                                 *
 *                                                                   *
 *        CASE (ACTION)                                              *
 *                                                                   *
 *        SUBCASE ('L')                                              *
 *              IF LASTNAME IS '*' THEN                              *
 *                 LIST ALL EMPLOYEES                                *
 *                 PREPARE OUTPUT_MESSAGE                            *
 *                 CALL DLI ISRT OUTPUT_MESSAGE                      *
 *               ELSE                                                *
 *                 IF LASTNAME CONTAINS '%' THEN                     *
 *                   LIST EMPLOYEES GENERIC                          *
 *                   PREPARE OUTPUT_MESSAGE                          *
 *                   CALL DLI ISRT OUTPUT_MESSAGE                    *
 *                 ELSE                                              *
 *                   LIST EMPLOYEES SPECIFIC                         *
 *                   PREPARE OUTPUT_MESSAGE                          *
 *                   CALL DLI ISRT OUTPUT_MESSAGE                    *
 *         ENDSUB                                                    *
 *                                                                   *
 *         SUBCASE ('U')                                             *
 *               DO WHILE INPUT PHONE_NO ^= BLANK                    *
 *                  UPDATE PHONE_NO FOR EMPLOYEE                     *
 *               END                                                 *
 *               PREPARE OUTPUT_MESSSAGE                             *
 *               CALL DLI ISRT OUTPUT MESSAGE                        *
 *         OTHERWISE                                                 *
 *               UNFORMATTED SCREEN                                  *
 *               PREPARE OUTPUT_MESSSAGE                             *
 *               CALL DLI ISRT OUTPUT MESSAGE                        *
 *         ENDSUB                                                    *
 *                                                                   *
 *       ENDCASE                                                     *
 *                                                                   *
 *         IF SQL OR DL/I ERROR HAS OCCURRED THEN                    *
 *            ROLLBACK                                               *
 *         PGM-STOP.                                                 *
 *      END.                                                         *
 *                                                                   *
 *------------------------------------------------------------------*/
1/*******************************************************************/
 /*       DECLARATION FOR INPUT:  MODNAME DSN8IPNI/DSN8IPLI          */
 /*******************************************************************/
0DCL  1 IN_MESSAGE       STATIC,
        2 LL             BIN FIXED (31),
        2 ZZ             CHAR (2),
        2 TC_CODE        CHAR (7),
        2 ACTION         CHAR (1),
        2 MESSAGE        CHAR (500);
0DCL  1 INPUT_1          BASED(ADDR(MESSAGE)),
        2 LNAME          CHAR (15),
        2 FNAME          CHAR (12);
0DCL  1 INPUT_2 (15)     BASED(ADDR(MESSAGE)),
        2 NEWNO          CHAR (4),
        2 ENO            CHAR (6);
-/*******************************************************************/
 /*        DECLARATION FOR OUTPUT: MODNAME DSN8IPNO/DSN8IPLO         */
 /*******************************************************************/
```

```
0DCL  1 OUT_MESSAGE      STATIC,
        2 LL             BIN FIXED (31),
        2 ZZ             BIN FIXED (15) INIT (0),
        2 ERROR          CHAR (79),
        2 OUTPUT         CHAR (1095);
0DCL  1 OUTPUT_1         BASED(ADDR(OUTPUT)),
        2 LASTNAME       CHAR (15),
        2 FIRSTNAME      CHAR (12);
0DCL  1 OUTPUT_2 (15)    BASED(ADDR(OUTPUT)),
        2 FIRSTNAME      CHAR (12),
        2 MIDDLEINITIAL  CHAR (1),
        2 LASTNAME       CHAR (15),
        2 PHONENUMBER    CHAR (4),
        2 EMPLOYEENUMBER CHAR (6),
        2 DEPTNUMBER     CHAR (3),
        2 DEPTNAME       CHAR (32);  /* 32 TO FIT ON ONE LINE */
 DCL  CHAR_SQLCODE CHAR (14);
 DCL 1 CHAR_SQLSTR BASED(ADDR(CHAR_SQLCODE)),
     2 CHAR_BLNK   CHAR(4),
     2 CHAR_SQLCOD CHAR(10);
0/*******************************************************************/
 /*        DECLARATION FOR IO / ALTPCB MASK                        */
 /*******************************************************************/
0DCL  (IOPCB_ADDR,ALTPCB_ADDR) POINTER;
0DCL  1 IOPCB        BASED (IOPCB_ADDR),
        2 IOLTERM    CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2);
0DCL  1 ALTPCB       BASED (ALTPCB_ADDR),
        2 ALTLTERM   CHAR (8),
        2 FILLER     CHAR (2),
        2 STC_CODE   CHAR (2);
0/*******************************************************************/
 /*        DECLARATION FOR PGM-LOGIC                               */
 /*******************************************************************/
0DCL  ONE          BIN FIXED (31) INIT (1)  STATIC;
 DCL  THREE        BIN FIXED (31) INIT (3)  STATIC;
 DCL  FOUR         BIN FIXED (31) INIT (4)  STATIC;
0DCL  GU_FKT       CHAR (4) INIT ('GU  ') STATIC;
 DCL  ISRT_FKT     CHAR (4) INIT ('ISRT') STATIC;
 DCL  CHNG_FKT     CHAR (4) INIT ('CHNG') STATIC;
 DCL  ROLL_FKT     CHAR (4) INIT ('ROLL') STATIC;
0DCL  MODNAME      CHAR (8) STATIC;
 DCL  FIRST        BIT  (1) STATIC;
 DCL  EMPLOYEE_NO  CHAR (6) STATIC;
 DCL  PHONE_NO     CHAR (4) STATIC;
0DCL  (I,M,ITAB) BIN FIXED(15);
0DCL  (ADDR,INDEX,SUBSTR) BUILTIN;
0DCL  TRANSLATE BUILTIN;
0DCL  SYSPRINT EXTERNAL PRINT FILE;
0DCL  PLITDLI EXTERNAL ENTRY;
0DCL  DSN8MPG EXTERNAL ENTRY;

     /****************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE      */
     /****************************************************/

 DCL  MODULE            CHAR (07) INIT ('DSN8IP3');
 DCL  OUTMSG            CHAR (69);

1/*******************************************************************/
 /*                     DECLARATION FOR SQL                        */
 /*******************************************************************/
0EXEC SQL INCLUDE SQLCA;           /* SQL COMMUNICATION AREA        */
                                   /* SQL DECLARATION FOR VIEW PHONE */
 EXEC SQL DECLARE VPHONE TABLE
        (LASTNAME       VARCHAR(15)         ,
         FIRSTNAME      VARCHAR(12)         ,
         MIDDLEINITIAL     CHAR(1)          ,
         PHONENUMBER       CHAR(4)          ,
         EMPLOYEENUMBER    CHAR(6)          ,
         DEPTNUMBER        CHAR(3)  NOT NULL,
         DEPTNAME       VARCHAR(36) NOT NULL);
                                   /* STUCTURE FOR PHONE RECORD      */
 DCL 1 PPHONE,
       2 LASTNAME         CHAR (15) VAR,
       2 FIRSTNAME        CHAR (12) VAR,
       2 MIDDLEINITIAL    CHAR (1),
       2 PHONENUMBER      CHAR (4),
       2 EMPLOYEENUMBER   CHAR (6),
       2 DEPTNUMBER       CHAR (3),
       2 DEPTNAME         CHAR (36) VAR;
```

```
                                     /* SQL DECLARATION FOR VIEW VEMPLP*/
 EXEC SQL DECLARE VEMPLP TABLE
        (EMPLOYEENUMBER    CHAR(6)          ,
         PHONENUMBER       CHAR(4));

                                     /* STRUCTURE FOR PEMPLP RECORD    */
 DCL 1 PEMP,
       2 EMPLOYEENUMBER   CHAR (6),
       2 PHONENUMBER      CHAR (4);
1/*******************************************************************/
 /*         SQL CURSORS                                             */
 /*******************************************************************/

 EXEC SQL DECLARE TELE1 CURSOR FOR
          SELECT *
          FROM VPHONE;

 EXEC SQL DECLARE TELE2 CURSOR FOR
          SELECT *
          FROM  VPHONE
          WHERE LASTNAME LIKE :INPUT_1.LNAME
          AND  FIRSTNAME LIKE :INPUT_1.FNAME;

 EXEC SQL DECLARE TELE3 CURSOR FOR
          SELECT *
          FROM  VPHONE
          WHERE LASTNAME  =    :INPUT_1.LNAME
          AND   FIRSTNAME LIKE :INPUT_1.FNAME;

1/*******************************************************************/
 /*         SQL RETURN CODE HANDLING                                */
 /*******************************************************************/

 EXEC SQL WHENEVER SQLERROR   GOTO P3_DBERROR;
 EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;
 EXEC SQL WHENEVER NOT FOUND  CONTINUE;

 /*******************************************************************/
 /*                  MAIN PROGRAM ROUTINE                           */
 /*******************************************************************/
                                                 /*INITIALIZATIONS */
0P3_START:
    IN_MESSAGE     = '';                          /* SCREEN INPUT  */
    OUT_MESSAGE    = '';                          /* SCREEN OUTPUT */
    OUT_MESSAGE.LL = 83;                          /* LINE LENGTH   */
    MODNAME        = 'DSN8IPNO';                  /* MODULE NAME   */
    FIRST          = '1'B;
    ITAB           = 0;                           /* COUNTER       */
0   CALL PLITDLI (THREE,GU_FKT,IOPCB,IN_MESSAGE);

                                          /*  IF INVALID DL/I    */
                                          /* STC-CODE ON GU MSG  */
                                          /*  PRINT ERROR MESSAGE*/
0   IF IOPCB.STC_CODE ^= ' ' THEN
     DO;
      CALL DSN8MPG (MODULE, '064E', OUTMSG);
      ERROR = OUTMSG||IOPCB.STC_CODE;
      CALL P3_SEND;
     END;

 /*******************************************************************/
 /*                  SELECT ACTION                                  */
 /*******************************************************************/
0   SELECT (ACTION);
      WHEN ('L') DO;                         /* ACTION - LIST          */

        /***********************************************************/
        /* REDISPLAY SELECTION SCREEN IF NO CRITERIA ENTERED          */
        /***********************************************************/
0       IF INPUT_1.LNAME = ' ' &
           INPUT_1.FNAME = ' ' THEN
          DO;
            CALL P3_SEND;
            GOTO P3_END;
          END;

        MODNAME = 'DSN8IPLO';                /* SELECT "LISTING" PANEL */

 /*******************************************************************/
 /*                  LIST ALL EMPLOYEES                             */
 /*******************************************************************/
0       IF INPUT_1.LNAME = '*' THEN        /* LIST ALL EMPLOYEES */
```

```
         DO;

           EXEC SQL OPEN TELE1;                /* OPEN CURSOR */

           EXEC SQL FETCH TELE1             /* GET FIRST RECORD   */
                   INTO :PPHONE;

0         IF SQLCODE = 100 THEN            /* NO EMPLOYEES FOUND */
           DO;                             /* PRINT ERROR MESSAGE*/
             MODNAME = 'DSN8IPNO';
             CALL DSN8MPG (MODULE, '008I', OUTMSG);
             ERROR = OUTMSG;
             CALL P3_SEND;
             GOTO P3_SELECT_20;
           END;

          CALL P3_PREPARE_SCREEN;          /* LIST FIRST EMPLOYEE*/

 P3_SELECT_10:
          EXEC SQL FETCH TELE1             /* GET NEXT RECORD  */
                   INTO :PPHONE;

0         IF SQLCODE = 100 THEN             /* FINISHED ?       */
           DO;
             ERROR = '';
             CALL P3_SEND;
             GOTO P3_SELECT_20;
           END;

          CALL P3_PREPARE_SCREEN;               /* LIST EMPLOYEE   */
          GOTO P3_SELECT_10;                      /* CONTINUE      */
 P3_SELECT_20:
          EXEC SQL CLOSE TELE1;                 /* CLOSE CURSOR    */
          GOTO P3_END;
         END;                                           /* END IF  */
 /**********************************************************************/
 /*                  LIST GENERIC EMPLOYEES                          */
 /**********************************************************************/
        ELSE DO;                        /* SELECT EMPLOYEES BY NAME*/
                                        /* SEARCH ON PART OF NAME? */
          IF INPUT_1.LNAME = '               ' THEN /* IS NAME BLANK*/
             INPUT_1.LNAME = '%%%%%%%%%%%%%%%';     /* SET PATTERN  */
          IF INDEX(INPUT_1.LNAME,'%') > 0 THEN
           DO;
                                        /* YES, SEARCH ON          */
                                        /* PART OF LAST NAME       */
                                        /* (OPT. PART FIRST NAME)  */
                                        /* TRANSLATE IT            */
            INPUT_1.LNAME = TRANSLATE(INPUT_1.LNAME,'%',' ');
            INPUT_1.FNAME = TRANSLATE(INPUT_1.FNAME,'%',' ');

            EXEC SQL OPEN TELE2;         /* OPEN CURSOR             */

            EXEC SQL FETCH TELE2         /* GET FIRST RECORD        */
                     INTO :PPHONE;

0           IF SQLCODE = 100 THEN        /* NO EMPLOYEES FOUND      */
             DO;                         /* PRINT ERROR MESSAGE     */
               MODNAME = 'DSN8IPNO';
               CALL DSN8MPG (MODULE, '008I', OUTMSG);
               ERROR = OUTMSG;
               CALL P3_SEND;
               GOTO P3_SELECT_40;
             END;

            CALL P3_PREPARE_SCREEN;        /* LIST FIRST EMPLOYEE    */

 P3_SELECT_30:
            EXEC SQL FETCH TELE2           /* GET NEXT RECORD        */
                     INTO :PPHONE;

0           IF SQLCODE = 100 THEN          /* FINISHED?              */
             DO;
               ERROR = '';
               CALL P3_SEND;
               GOTO P3_SELECT_40;
             END;

            CALL P3_PREPARE_SCREEN;        /* LIST EMPLOYEE          */
            GOTO P3_SELECT_30;             /* CONTINUE               */

 P3_SELECT_40:
```

```
            EXEC SQL CLOSE TELE2;          /* CLOSE CURSOR          */
            GOTO P3_END;
           END;                            /*    END IF             */
 /*********************************************************************/
 /*                   LIST SPECIFIC EMPLOYEE(S)                       */
 /*********************************************************************/

          ELSE DO;                        /* NO - SEARCH ON LAST NAME*/
                                          /*AND OPTIONALLY FIRST NAME*/
             INPUT_1.FNAME = TRANSLATE(INPUT_1.FNAME,'%',' ');

             EXEC SQL OPEN TELE3;          /* OPEN CURSOR         */

             EXEC SQL FETCH TELE3          /* GET FIRST RECORD    */
                     INTO :PPHONE;

0            IF SQLCODE = 100 THEN         /* EMPLOYEE NOT FOUND  */
              DO;                          /* PRINT ERROR MESSAGE */
                MODNAME = 'DSN8IPNO';
                CALL DSN8MPG (MODULE, '008I', OUTMSG);
                ERROR = OUTMSG;
                CALL P3_SEND;
                GOTO P3_SELECT_60;
              END;

             CALL P3_PREPARE_SCREEN;      /* LIST FIRST EMPLOYEE  */

 P3_SELECT_50:
             EXEC SQL FETCH TELE3          /* GET NEXT RECORD     */
                     INTO :PPHONE;

0           IF SQLCODE = 100 THEN          /* FINISHED ?          */
             DO;
               ERROR = '';
               CALL P3_SEND;
               GOTO P3_SELECT_60;
             END;

            CALL P3_PREPARE_SCREEN;        /* LIST EMPLOYEE          */
            GOTO P3_SELECT_50;             /* CONTINUE               */

 P3_SELECT_60:
            EXEC SQL CLOSE TELE3;          /* CLOSE CURSOR           */
          END;                             /* END IF                 */
        END;                               /* END IF                 */
      END;                                 /* END WHEN               */
                                          /* CHANGE ERROR HANDLING   */
                                          /* FOR UPDATE              */
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      EXEC SQL WHENEVER SQLWARNING CONTINUE;
 /*********************************************************************/
 /*                   UPDATE PHONE NUMBERS FOR EMPLOYEES              */
 /*********************************************************************/
0     WHEN ('U') DO;                       /* TELEPHONE UPDATE       */

        OUT_MESSAGE.LL = 110;
        MODNAME        = 'DSN8IPNO';
                                          /* FIND WHICH NUMBERS HAVE */
                                          /* BEEN UPDATED            */
        ERROR = '';                       /* SET IN CASE NO UPDATES  */
0       DO I = 1 TO 15;
          IF INPUT_2.NEWNO(I) = ' ' THEN; /* NO UPDATE ON THIS LINE  */

          ELSE DO;                        /* PERFORM UPDATE          */
            EMPLOYEE_NO = INPUT_2.ENO(I);
            PHONE_NO    = INPUT_2.NEWNO(I);
0           EXEC SQL UPDATE VEMPLP
                     SET PHONENUMBER = :PHONE_NO
                     WHERE EMPLOYEENUMBER = :EMPLOYEE_NO;

                                          /* UPDATE SUCCESSFUL       */
                                          /* PRINT CONFIRMATION      */
                                          /* MESSAGE                 */
0           IF SQLCODE = 0 THEN
             DO;
0              CALL DSN8MPG (MODULE, '004I', OUTMSG);
               ERROR = OUTMSG;
             END;
                                          /* UPDATE FAILED           */
                                          /* PRINT ERROR MESSAGE     */
            ELSE
             DO;
```

```
                CALL DSN8MPG (MODULE, '007E', OUTMSG);
                ERROR = OUTMSG;
                GOTO P3_DBERROR2;
              END;

           END;                                     /*   END IF       */
         END;                                       /*   END WHEN     */
0        CALL P3_SEND;
       END;

0      OTHERWISE                            /* UNFORMATTED SCREEN     */
        DO;
          OUT_MESSAGE.LL = 110;
          MODNAME        = 'DSN8IPNO';
          CALL P3_SEND;
        END;
     END;                                   /* END SELECT             */
0    GOTO P3_END;
1/*******************************************************************/
 /*        SQL ERROR HANDLING                                       */
 /*******************************************************************/
0P3_DBERROR:
    CALL DSN8MPG (MODULE, '060E', OUTMSG);
    CHAR_SQLCODE = SQLCODE;
    ERROR = OUTMSG||CHAR_SQLCOD;
    PUT DATA (ERROR,SQLWARN0);     /*PRINT ERROR MESSAGE */

0P3_DBERROR2:
0   CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM);

0   IF ALTPCB.STC_CODE ^= ' ' THEN;     /* PERFORM ROLLBACK  */
    ELSE CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);
0         CALL PLITDLI (ONE,ROLL_FKT);
0   GOTO P3_END;

1/*******************************************************************/
 /*        PRINT INFORMATION ON SCREEN                              */
 /*******************************************************************/
1P3_PREPARE_SCREEN:
    PROC;
                                       /*IF ANOTHER PAGE */
    IF ITAB = 15 THEN                  /* PRINT SCROLLING MESSAGE */
     DO;
       CALL DSN8MPG (MODULE, '058I', OUTMSG);
       ERROR = OUTMSG;
       CALL P3_SEND;
       ITAB           = 0;                /* INITIALIZE COUNTER      */
       OUT_MESSAGE.LL = 83;               /* INITIALIZE LINE LENGTH  */
     END;

    ITAB = ITAB + 1;                      /* INCREMENT COUNTER       */
                                          /* MOVE DATA TO OUTPUT AREA*/
    OUTPUT_2 (ITAB) = PPHONE , BY NAME;
    OUT_MESSAGE.LL  = OUT_MESSAGE.LL + 73;
    RETURN;

 END P3_PREPARE_SCREEN;
1P3_SEND:
    PROC;

    IF FIRST THEN
     DO;
       CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME);
       FIRST = '0'B;
     END;

    ELSE  CALL PLITDLI (THREE,ISRT_FKT,IOPCB,OUT_MESSAGE);

0   IF IOPCB.STC_CODE = ' ' THEN RETURN;

                                /* INVALID DL/I STC-CODE ON ISRT MSG*/
                                /* PRINT ERROR MESSAGE */
    CALL DSN8MPG (MODULE, '065E', OUTMSG);
0   ERROR = OUTMSG||IOPCB.STC_CODE;
0   CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM);

0   IF ALTPCB.STC_CODE ^= ' ' THEN;           /* PERFORM ROLLBACK */
      ELSE CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);
0        CALL PLITDLI (ONE,ROLL_FKT);
    RETURN;
 /****************************************************************/
0END P3_SEND;                                /* END OF PROGRAM */
```

```
0P3_END:
    END DSN8IP3;
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSNTEJ4C

THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH COBOL.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ4C                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 4                                         00050000
//*                     COBOL, IMS                                      00060000
//*                                                                     00070000
//*    LICENSED MATERIALS - PROPERTY OF IBM                             00080000
//*    5635-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.         00100000
//*                                                                     00110000
//*    STATUS = VERSION 11                                              00120001
//*                                                                     00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE      00140000
//*             APPLICATIONS AT SITES WITH COBOL.  IT PREPARES THE      00150000
//*             COBOL IMS PROGRAM.                                      00160000
//*                                                                     00170000
//*             RUN THIS JOB ANYTIME AFTER PHASE 2.                     00180000
//*                                                                     00190000
//*  CHANGE ACTIVITY =                                                  00200000
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651 00201000
//*               ADD SET CURRENT SQLID                                 00202002
//*                                                                     00210000
//********************************************************************* 00220000
//*                                                                     00230000
//JOBLIB  DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                              00240000
//*        STEP  1: PREPARE SQL 0 PART OF PROGRAM                       00250000
//PH04CS01 EXEC DSNHICOB,                                               00260000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00270000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00280000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),       00290000
//         PARM.LKED='XREF,NCAL',                                       00300000
//         MEM=DSN8IC0                                                  00310000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IC0),                     00320000
//             DISP=SHR                                                 00330000
//PC.SYSCIN    DD DSN=&&DSNHOUT0                                        00340000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00350000
//             DISP=SHR                                                 00360000
//             DD DSN=DSN!!0.SDSNSAMP,                                  00370000
//             DISP=SHR                                                 00380000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8IC0),                         00390000
//             DISP=SHR                                                 00400000
//COB.SYSIN    DD DSN=&&DSNHOUT0                                        00410000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IC0),                      00420000
//             DISP=SHR                                                 00430000
//*                                                                     00440000
//*        STEP  2: PREPARE SQL 1 PART OF PROGRAM                       00450000
//PH04CS02 EXEC DSNHICOB,                                               00460000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00470000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00480000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),       00490000
//         PARM.LKED='XREF,NCAL',                                       00500000
//         COND=(4,LT),                                                 00510000
//         MEM=DSN8IC1                                                  00520000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IC1),                     00530000
//             DISP=SHR                                                 00540000
//PC.SYSCIN    DD DSN=&&DSNHOUT1                                        00550000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00560000
//             DISP=SHR                                                 00570000
//             DD DSN=DSN!!0.SDSNSAMP,                                  00580000
//             DISP=SHR                                                 00590000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8IC1),                         00600000
//             DISP=SHR                                                 00610000
//COB.SYSIN    DD DSN=&&DSNHOUT1                                        00620000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IC1),                      00630000
//             DISP=SHR                                                 00640000
```

```
//*                                                                     00650000
//*        STEP  3: PREPARE SQL 2 PART OF PROGRAM                       00660000
//PH04CS03 EXEC DSNHICOB,                                               00670000
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,               00680000
//         NOXREF,'SQL(DB2)','DEC(31)'),                                00690000
//         PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),       00700000
//         PARM.LKED='XREF,NCAL',                                       00710000
//         COND=(4,LT),                                                 00720000
//         MEM=DSN8IC2                                                  00730000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IC2),                     00740000
//             DISP=SHR                                                 00750000
//PC.SYSCIN    DD DSN=&&DSNHOUT2                                        00760000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00770000
//             DISP=SHR                                                 00780000
//             DD DSN=DSN!!0.SDSNSAMP,                                  00790000
//             DISP=SHR                                                 00800000
//PC.SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8IC2),                         00810000
//             DISP=SHR                                                 00820000
//COB.SYSIN    DD DSN=&&DSNHOUT2                                        00830000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IC2),                      00840000
//             DISP=SHR                                                 00850000
//*                                                                     00860000
//*        STEP  4: LINKEDIT THE ENTIRE PROGRAM                         00870000
//PH04CS04 EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(4,LT)               00880000
//SYSLIB   DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED                          00890000
//         DD DISP=SHR,DSN=DSN!!0.SDSNLOAD                              00900000
//         DD DISP=SHR,DSN=IMSVS.RESLIB                                 00910000
//SYSLIN   DD DDNAME=SYSIN                                              00920000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           00930000
//SYSPRINT DD SYSOUT=*                                                  00940000
//SYSUDUMP DD SYSOUT=*                                                  00950000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(50,50))                           00960000
//SYSIN    DD *                                                         00970000
  INCLUDE SYSLIB(DFSLI000)                                              00980000
  INCLUDE SYSLMOD(DSN8IC0)                                              00990000
  INCLUDE SYSLMOD(DSN8IC1)                                              01000000
  INCLUDE SYSLMOD(DSN8IC2)                                              01010000
  INCLUDE SYSLMOD(DSN8MCG)                                              01020000
  ENTRY   DLITCBL                                                       01030000
  NAME    DSN8IC0(R)                                                    01040000
//*                                                                     01050000
//*        STEP  5: BIND THE PLAN                                       01060000
//PH04CS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    01070000
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                          01080000
//SYSUDUMP DD SYSOUT=*                                                  01090000
//SYSTSPRT DD SYSOUT=*                                                  01100000
//SYSPRINT DD SYSOUT=*                                                  01110000
//SYSIN   DD *                                                          01120000
   SET CURRENT SQLID = 'SYSADM';                                        01121002
   GRANT BIND, EXECUTE ON PLAN DSN8IC0                                  01130003
     TO PUBLIC;                                                         01131003
//SYSTSIN  DD *                                                         01140000
DSN SYSTEM(DSN)                                                         01150000
BIND PACKAGE (DSN8IC!!) MEMBER(DSN8IC1) -                               01160000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 01170000
BIND PACKAGE (DSN8IC!!) MEMBER(DSN8IC2) -                               01180000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 01190000
BIND PLAN(DSN8IC0) PKLIST(DSN8IC!!.*) -                                 01200000
     ACTION(REPLACE) RETAIN +                                           01201000
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          01210000
RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  01220000
      LIB('DSN!!0.RUNLIB.LOAD')                                         01230000
END                                                                     01240000
//*                                                                     01250000
//*        STEP  6: CREATE THE MFS MAPS                                 01260000
//PH04CS06 EXEC MFSUTL,COND=(4,LT)                                      01270000
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPG),                                01280000
//             DISP=SHR                                                 01290000
//*                                                                     01300000
//*        STEP  7: CREATE THE MFS MAPS                                 01310000
//PH04CS07 EXEC MFSUTL,COND=(4,LT)                                      01320000
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPD),                                01330000
//             DISP=SHR                                                 01340000
//*                                                                     01350000
//*        STEP  8: RUN THE PSBGEN                                      01360000
//PH04CS08 EXEC PSBGEN,MBR=DSN8IC0,COND=(4,LT)                          01370000
//C.SYSIN DD *                                                          01380000
         PRINT NOGEN                                                    01390000
         PCB   TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES   01400000
         PSBGEN PSBNAME=DSN8IC0,LANG=COBOL                              01410000
         END                                                            01420000
//*                                                                     01430000
```

```
//*        STEP  9: RUN THE ACBGEN
//PH04CS09 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN DD *
 BUILD PSB=DSN8IC0
//*
//*        ALSO ADD MEMBER DSN8FIMS TO THE SYSDEF TO ADD TRANSACTIONS
//*
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

DSNTEJ4P

THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH PL/I.

```
//*********************************************************************
//*  NAME = DSNTEJ4P
//*
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//*                     PHASE 4
//*                     PL/I, IMS
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5605-DB2
//*    (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 11
//*
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE
//*             APPLICATIONS AT SITES WITH PL/I.  IT PREPARES THE
//*             PL/I IMS PROGRAM.
//*
//*             RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//*  CHANGE ACTIVITY =
//*    11/07/2012 ADD RETAIN TO BIND PLAN STMTS   DN1651_INST1 / DN1651
//*               ADD SET CURRENT SQLID
//*
//*********************************************************************
//*
//JOBLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//*        STEP  1: PREPARE SQL 0 PART OF PROGRAM
//PH04PS01 EXEC DSNHPLI,MEM=DSN8IP0,
//          PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
//          PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
//          PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
//          'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
//          PARM.LKED='NCAL'
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP0),
//             DISP=SHR
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP0),
//             DISP=SHR
//PC.SYSCIN    DD DSN=&&DSNHOUT0
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//             DD DSN=DSN!!0.SDSNSAMP,
//             DISP=SHR
//PLI.SYSIN    DD DSN=&&DSNHOUT0
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP0),
//             DISP=SHR
//*
//*        STEP  2: PREPARE SQL 1 PART OF PROGRAM
//PH04PS02 EXEC DSNHPLI,MEM=DSN8IP1,
//          COND=(4,LT),
//          PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
//          PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
//          PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
//          'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
//          PARM.LKED='NCAL'
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP1),
//             DISP=SHR
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP1),
//             DISP=SHR
//PC.SYSCIN    DD DSN=&&DSNHOUT1
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//             DD DSN=DSN!!0.SDSNSAMP,
```

```
//             DISP=SHR                                                 00610000
//PLI.SYSIN    DD DSN=&&DSNHOUT1                                        00620000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP1),                      00630000
//             DISP=SHR                                                 00640000
//*                                                                     00650000
//*        STEP  3: PREPARE SQL 2 PART OF PROGRAM                       00660000
//PH04PS03 EXEC DSNHPLI,MEM=DSN8IP2,                                    00670000
//           COND=(4,LT),                                               00680000
//           PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      00690000
//           PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',                  00700000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',     00710000
//           'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),               00720000
//           PARM.LKED='NCAL'                                           00730000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP2),                         00740000
//             DISP=SHR                                                 00750000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP2),                     00760000
//             DISP=SHR                                                 00770000
//PC.SYSCIN    DD DSN=&&DSNHOUT2                                        00780000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               00790000
//             DISP=SHR                                                 00800000
//             DD DSN=DSN!!0.SDSNSAMP,                                  00810000
//             DISP=SHR                                                 00820000
//PLI.SYSIN    DD DSN=&&DSNHOUT2                                        00830000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP2),                      00840000
//             DISP=SHR                                                 00850000
//*                                                                     00860000
//*        STEP  4: PREPARE TELEPHONE PROGRAM                           00870000
//PH04PS04 EXEC DSNHPLI,MEM=DSN8IP3,                                    00880000
//           COND=(4,LT),                                               00890000
//           PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      00900000
//           PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',                  00910000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            00920000
//           'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),               00930000
//           PARM.LKED='NCAL'                                           00940000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP3),                         00950000
//             DISP=SHR                                                 00960000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP3),                     00970000
//             DISP=SHR                                                 00980000
//PC.SYSCIN    DD DSN=&&DSNHOUT3                                        00990000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01000000
//             DISP=SHR                                                 01010000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01020000
//             DISP=SHR                                                 01030000
//PLI.SYSIN    DD DSN=&&DSNHOUT3                                        01040000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP3),                      01050000
//             DISP=SHR                                                 01060000
//*                                                                     01070000
//*        STEP  5: PREPARE SQL 0 PART OF PROJECT APPLICATION           01080000
//PH04PS05 EXEC DSNHPLI,MEM=DSN8IP6,                                    01090000
//           COND=(4,LT),                                               01100000
//           PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      01110000
//           PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',                  01120000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            01130000
//           'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),               01140000
//           PARM.LKED='NCAL'                                           01150000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP6),                         01160000
//             DISP=SHR                                                 01170000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP6),                     01180000
//             DISP=SHR                                                 01190000
//PC.SYSCIN    DD DSN=&&DSNHOUT6                                        01200000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01210000
//             DISP=SHR                                                 01220000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01230000
//             DISP=SHR                                                 01240000
//PLI.SYSIN    DD DSN=&&DSNHOUT6                                        01250000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP6),                      01260000
//             DISP=SHR                                                 01270000
//*                                                                     01280000
//*        STEP  6: PREPARE SQL 1 PART OF PROGRAM                       01290000
//PH04PS06 EXEC DSNHPLI,MEM=DSN8IP7,                                    01300000
//           COND=(4,LT),                                               01310000
//           PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      01320000
//           PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',                  01330000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',     01340000
//           'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),               01350000
//           PARM.LKED='NCAL'                                           01360000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP7),                         01370000
//             DISP=SHR                                                 01380000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP7),                     01390000
//             DISP=SHR                                                 01400000
//PC.SYSCIN    DD DSN=&&DSNHOUT7                                        01410000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01420000
```

```
//             DISP=SHR                                                 01430000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01440000
//             DISP=SHR                                                 01450000
//PLI.SYSIN    DD DSN=&&DSNHOUT7                                        01460000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP7),                      01470000
//             DISP=SHR                                                 01480000
//*                                                                     01490000
//*        STEP  7: PREPARE SQL 2 PART OF PROGRAM                       01500000
//PH04PS07 EXEC DSNHPLI,MEM=DSN8IP8,                                    01510000
//           COND=(4,LT),                                               01520000
//           PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      01530000
//           PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',                  01540000
//           PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',     01550000
//           'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),               01560000
//           PARM.LKED='NCAL'                                           01570000
//PPLI.SYSIN   DD DSN=DSN!!0.SDSNSAMP(DSN8IP8),                         01580000
//             DISP=SHR                                                 01590000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8IP8),                     01600000
//             DISP=SHR                                                 01610000
//PC.SYSCIN    DD DSN=&&DSNHOUT8                                        01620000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01630000
//             DISP=SHR                                                 01640000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01650000
//             DISP=SHR                                                 01660000
//PLI.SYSIN    DD DSN=&&DSNHOUT8                                        01670000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP8),                      01680000
//             DISP=SHR                                                 01690000
//*                                                                     01700000
//*        STEP  8: LINKEDIT PROGRAM TOGETHER                           01710000
//PH04PS08 EXEC PGM=IEWL,PARM='LIST,XREF,LET',                          01720000
//         COND=(4,LT)                                                  01730000
//SYSLIB   DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED                          01740000
//         DD DISP=SHR,DSN=DSN!!0.SDSNLOAD                              01750000
//         DD DISP=SHR,DSN=IMSVS.RESLIB                                 01760000
//SYSLIN   DD DDNAME=SYSIN                                              01770000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           01780000
//SYSPRINT DD SYSOUT=*                                                  01790000
//SYSUDUMP DD SYSOUT=*                                                  01800000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(50,50))                           01810000
//SYSIN    DD *                                                         01820000
  INCLUDE SYSLIB(DFSLI000)                                              01830000
  INCLUDE SYSLMOD(DSN8IP0)                                              01840000
  INCLUDE SYSLMOD(DSN8IP1)                                              01850000
  INCLUDE SYSLMOD(DSN8IP2)                                              01860000
  INCLUDE SYSLMOD(DSN8MPG)                                              01870000
  ENTRY   CEESTART                                                      01880000
  NAME    DSN8IP0(R)                                                    01890000
  INCLUDE SYSLIB(DFSLI000)                                              01900000
  INCLUDE SYSLMOD(DSN8IP3)                                              01910000
  INCLUDE SYSLMOD(DSN8MPG)                                              01920000
  ENTRY   CEESTART                                                      01930000
  NAME    DSN8IH0(R)                                                    01940000
  INCLUDE SYSLIB(DFSLI000)                                              01950000
  INCLUDE SYSLMOD(DSN8IP6)                                              01960000
  INCLUDE SYSLMOD(DSN8IP7)                                              01970000
  INCLUDE SYSLMOD(DSN8IP8)                                              01980000
  INCLUDE SYSLMOD(DSN8MPG)                                              01990000
  ENTRY   CEESTART                                                      02000000
  NAME    DSN8IQ0(R)                                                    02010000
//*                                                                     02020000
//*        STEP  9: BIND PROGRAMS                                       02030000
//PH04PS09 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    02040000
//DBRMLIB  DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                          02050000
//SYSUDUMP DD SYSOUT=*                                                  02060000
//SYSTSPRT DD SYSOUT=*                                                  02070000
//SYSPRINT DD SYSOUT=*                                                  02080000
//SYSIN    DD *                                                         02090000
   SET CURRENT SQLID = 'SYSADM';                                        02091001
   GRANT BIND, EXECUTE ON PLAN DSN8IP0, DSN8IQ0, DSN8IH0                02100002
     TO PUBLIC;                                                         02101002
//SYSTSIN  DD *                                                         02110000
DSN SYSTEM(DSN)                                                         02120000
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP1) +                                02174000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 02175000
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP2) +                                02176000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 02177000
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP3) +                                02178000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 02179000
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP7) +                                02179300
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 02179400
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP8) +                                02179500
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 02179600
```

```
BIND PLAN(DSN8IP0) +
     PKLIST(DSN8IP!!.DSN8IP1, DSN8IP!!.DSN8IP2) +
     ACTION(REPLACE) RETAIN +
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IQ0) +
     PKLIST(DSN8IP!!.DSN8IP7, DSN8IP!!.DSN8IP8) +
     ACTION(REPLACE) RETAIN +
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IH0)  PKLIST(DSN8IP!!.DSN8IP3) +
     ACTION(REPLACE) RETAIN +
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
      LIB('DSN!!0.RUNLIB.LOAD')
END
//*
//*        STEP 10: CREATE MFS MAPS FOR ORGANIZATION APPLICATION
//PH04PS10 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPG),
//            DISP=SHR
//*
//*        STEP 11: CREATE MFS MAPS FOR ORGANIZATION APPLICATION
//PH04PS11 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPD),
//            DISP=SHR
//*
//*        STEP 12: CREATE MFS MAPS FOR PROJECT APPLICATION
//PH04PS12 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPF),
//            DISP=SHR
//*
//*        STEP 13: CREATE MFS MAPS FOR PROJECT APPLICATION
//PH04PS13 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPE),
//            DISP=SHR
//*
//*        STEP 14: CREATE MFS MAPS FOR TELEPHONE APPLICATION
//PH04PS14 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPL),
//            DISP=SHR
//*
//*        STEP 15: CREATE MFS MAPS FOR TELEPHONE APPLICATION
//PH04PS15 EXEC MFSUTL,COND=(4,LT)
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8IPN),
//            DISP=SHR
//*
//*        STEP 16: RUN PSBGEN
//PH04PS16 EXEC PSBGEN,MBR=DSN8IP0,COND=(4,LT)
//C.SYSIN  DD *
         PRINT NOGEN
         PCB   TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
         PSBGEN PSBNAME=DSN8IP0,LANG=PLI
         END
//*
//*        STEP 17: RUN ACBGEN
//PH04PS17 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN  DD *
 BUILD PSB=DSN8IP0
//*
//*        STEP 18: RUN PSBGEN
//PH04PS18 EXEC PSBGEN,MBR=DSN8IQ0,COND=(4,LT)
//C.SYSIN  DD *
         PRINT NOGEN
         PCB   TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
         PSBGEN PSBNAME=DSN8IQ0,LANG=PLI
         END
//*
//*        STEP 19 : RUN ACBGEN
//PH04PS19 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN  DD *
 BUILD PSB=DSN8IQ0
//*
//*        STEP 20 : RUN PSBGEN
//PH04PS20 EXEC PSBGEN,MBR=DSN8IH0,COND=(4,LT)
//C.SYSIN  DD *
         PRINT NOGEN
         PCB   TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
         PSBGEN PSBNAME=DSN8IH0,LANG=PLI
         END
//*
//*        STEP 21 : RUN ACBGEN
//PH04PS21 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN  DD *
```

```
 BUILD PSB=DSN8IH0                                                           02900000
//*                                                                          02910000
//*  ALSO ADD MEMBER DSN8FIMS TO THE SYSDEF, TO ADD TRANSACTIONS              02920000
//*                                                                          02930000
```

Related reference

"Sample applications in IMS" on page 1332
A set of Db2 sample applications run in the IMS environment.

Sample applications in CICS

A set of Db2 sample applications run in the CICS environment.

Table 184. Sample Db2 applications for CICS

Application	Program name	JCL member name	Description
Organization	DSN8CC0 DSN8CC1 DSN8CC2	DSNTEJ5C	CICS COBOL Organization Application
Organization	DSN8CP0 DSN8CP1 DSN8CP2	DSNTEJ5P	CICS PL/I Organization Application
Project	DSN8CP6 DSN8CP7 DSN8CP8	DSNTEJ5P	CICS PL/I Project Application
Phone	DSN8CP3	DSNTEJ5P	CICS PL/I Phone Application. This program lists employee telephone numbers and updates them if requested.

Related reference

Data sets that the precompiler uses
When you invoke the precompiler you need to provide data sets that contain input for the precompiler, such as the host programming statements and SQL statements. You also need to provide data sets where the precompiler can store its output, such as the modified source code and diagnostics messages.

DSN8CC0

THIS MODULE ISSUES CICS RECEIVE MAP TO RETRIEVE INPUT, CALLS DSN8CC1, AND ISSUE CICS SEND MAP AFTER RETURNING.

```
        IDENTIFICATION DIVISION.
       *------------------------
        PROGRAM-ID. DSN8CC0.

       **** DSN8CC0 - SUBSYSTEM INTERFACE MODULE FOR CICS/VS - COBOL ***
       *
       *   MODULE NAME = DSN8CC0
       *
       *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION
       *                      SUBSYSTEM INTERFACE MODULE
       *                      CICS
       *                      COBOL
       *
       *Licensed Materials - Property of IBM
       *5605-DB2
       *(C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.
       *
```

```
*STATUS = Version 10
*
*   FUNCTION =  THIS MODULE ISSUES CICS RECEIVE MAP TO RETRIEVE
*               INPUT, CALLS DSN8CC1, AND ISSUE CICS SEND MAP
*               AFTER RETURNING.
*
*   NOTES =
*       1. THIS IS A CICS PSEUDO CONVERSATION PROGRAM WHICH
*          INITIALIZES ITSELF WHEN A TERMINAL OPERATOR ENTERS
*          INPUT AFTER VIEWING THE SCREEN SENT BY PREVIOUS
*          ITERATIONS OF THE PROGRAM.
*
*      DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED:
*                              DSN8MCMG AND DSN8MCMD
*                     MODULE  DSN8CC1 IS REQUIRED.
*                     DCLGEN STRUCTURE DSN8MCCS IS REQUIRED
*                     INCLUDED COBOL STRUCTURE DSN8MCCA IS
*                     REQUIRED.
*
*      RESTRICTIONS = NONE
*
*
*   MODULE TYPE =
*      PROCESSOR   = DB2 PRECOMPILER,CICS TRANSLATOR,COBOL COMPIL
*      MODULE SIZE = SEE LINK-EDIT
*      ATTRIBUTES  = REUSABLE
*
*   ENTRY POINT = DSN8CC0
*      PURPOSE = SEE FUNCTION
*      LINKAGE = CICS/OS/VS ENTRY
*
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
*             SYMBOLIC LABEL/NAME = DSN8CCGI
*             DESCRIPTION = CICS/OS/VS BMS MAP FOR GENERAL INPUT
*
*             SYMBOLIC LABEL/NAME = DSN8CCDI
*             DESCRIPTION = CICS/OS/VS BMS MAP FOR DETAIL INPUT
*
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
*             SYMBOLIC LABEL/NAME = DSN8CCGO
*             DESCRIPTION = CICS/OS/VS BMS MAP FOR GENERAL OUTPUT
*
*             SYMBOLIC LABEL/NAME = DSN8CCDO
*             DESCRIPTION = CICS/OS/VS BMS MAP FOR DETAIL OUTPUT
*
*
*   EXIT-NORMAL = CICS RETURN TRANSID
*
*   EXIT-ERROR = SQL ERROR FOR SQL ERRORS
*                CICS ABEND FOR CICS PROBLEMS
*
*      RETURN CODE = NONE
*
*      ABEND CODES = LSCR - LOGICAL SCREEN SET INCORRECTLY
*
*      ERROR-MESSAGES = NONE
*
*
*   EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS
*     ROUTINES/SERVICES =
*         CICS/VS SERVICES
*         DSN8CC1            - SQL 1 MAINLINE CODE
*
*     DATA-AREAS  =
*         DSN8MCCA           - PARAMETER TO BE PASSED TO DSN8CC1
*                               COMMON AREA
*         DSN8MCCS           - DECLARE CONVERSATION STATUS
*         DSN8MCC2           - COMMON AREA PART 2
*         DSN8MCMD           - CICS/OS/VS COBOL MAP, ORGANIZATION
*         DSN8MCMG           - CICS/OS/VS COBOL MAP, ORGANIZATION
*
*     CONTROL-BLOCKS =
*         SQLCA               - SQL COMMUNICATION AREA
*
*   TABLES =  NONE
*
*   CHANGE-ACTIVITY =
*   - 10/18/2005  PK03311  INITIALIZE UNINITIALIZED STORAGE   @01
*
*  *PSEUDOCODE*
```

```
      *
      *     PROCEDURE
      *     DECLARATIONS.
      *        ALLOCATE COBOL WORK AREA FOR COMMAREA.
      *        PUT MODULE NAME 'DSN8CC0' IN AREA USED BY ERROR-HANDLER.
      *        PUT CICS EIBTRMID IN PCONVSTA.CONVID TO BE PASSED TO
      *        DSN8CC1.
      *        RETRIEVE LASTCR FROM VCONA USING THE CONVID TO DETERMINE
      *        WHICH OF THE TWO BMS MAPS SHOULD BE USED TO MAP IN DATA.
      *
      *
      *     IF RETRIEVAL OF MAPS IS SUCCESSFUL, THEN DO;
      *        EXEC CICS RECEIVE MAP ACCORDING TO SPECIFIED LASTSCR
      *
      *        IF MAPFAIL CONDITION IS RAISED* THEN DO;
      *                COMPARM.PFKIN = '00'
      *                GO TO CC0SEND
      *        END
      *
      *        ELSE
      *                PUT DATA FROM MAP INTO COMPARM **
      *
      *     ELSE
      *                IT IS A NEW CONVERSATION
      *                AND NO EXEC CICS RECEIVE MAP IS ISSUED.
      *
      *     CC0SEND:
      *     EXEC CICS LINK PROGRAM('DSN8CC1') COMMAREA(COMMAREA).
      *     UPON RETURN FROM DSN8CC1, EXEC CICS SEND MAP ACCORDING TO
      *          THE TYPE SPECIFIED IN PCONVSTA.LASTSCR.
      *     EXEC CICS RETURN TRANSID(D8CS).
      *
      *   END.
      *
      *    *    I.E. LAST CONVERSATION EXISTS, BUT OPERATOR HAD ENTERED
      *         DATA FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A
      *         FORMATTED SCREEN AND PRESSED ENTER
      *
      *    **   COMPARM.PFKIN = PF KEY ACTUALLY USED I.E. '01' FOR
      *           PF1 ..
      *---------------------------------------------------------------**


       ENVIRONMENT DIVISION.
      *---------------------

       DATA DIVISION.
      *--------------
       WORKING-STORAGE SECTION.
       77   FOUND           PIC  S99.
            EXEC SQL INCLUDE SQLCA    END-EXEC.
            EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
       01   COMMAREA.
            EXEC SQL INCLUDE DSN8MCCA END-EXEC.
            EXEC SQL INCLUDE DSN8MCCS END-EXEC.
            EXEC SQL INCLUDE DSN8MCMG END-EXEC.
            EXEC SQL INCLUDE DSN8MCMD END-EXEC.
      *****************************************************************-
      *     MAPD REDEFINES THE COBOL STRUCTURE ASSOCIATED WITH THE
      *     CICS MAP DSN8CCD.
      *****************************************************************-
       01   MAPD REDEFINES DSN8CCDI.
            02  FILLER      PIC X(387).
            02  SUBMAP      OCCURS 15 TIMES.
              03 COL1LEN    PIC S9(4) COMP.
              03 COL1ATTR   PIC X(1).
              03 COL1DATA   PIC X(37).
              03 COL2LEN    PIC S9(4) COMP.
              03 COL2ATTR   PIC X(1).
              03 COL2DATA   PIC X(40).
      *****************************************************************-
      *     PFKEYS IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT
      *     PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID.
      *****************************************************************-
       01   PFKEYS-DUMB.
         02 PFKEYS-ALL      PIC  X(24) VALUE '123456789:#@ABCDEFGHI>.<'.
         02 PFKEYS  REDEFINES PFKEYS-ALL PIC X(1)  OCCURS 24 TIMES.
      *****************************************************************-
      *     PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS
      *     ALLOWED AS INPUT TO DSN8CC1 AND DSN8CC2 ETC.
      *****************************************************************-
       01   PFK-DUMB.
```

```
    02 PFK-ALL        PIC  X(24) VALUE '010203040506070809101112'.
    02 PFK     REDEFINES PFK-ALL    PIC X(2)  OCCURS 12 TIMES.


  PROCEDURE DIVISION.
 *------------------
 *****************************************************************-
 *     SQL RETURN CODE HANDLING
 *****************************************************************-
      EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
      EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

 *****************************************************************-
 *     ALLOCATE COBOL WORK AREA / INITIALIZE VARIABLES
 *****************************************************************-
 * INIT AREA INCLUDED BY DSN8MCCA
      MOVE SPACES TO COMMAREA.
 * INIT AREA INCLUDED BY DSN8MCCS                              @01
      MOVE SPACES TO PCONA.
 * INIT AREA INCLUDED BY DSN8MCMG                              @01
      MOVE ZEROES TO ATITLEL  OF DSN8CCGI.
      MOVE SPACES TO ATITLEI  OF DSN8CCGI.
      MOVE ZEROES TO AMAJSYSL OF DSN8CCGI.
      MOVE SPACES TO AMAJSYSI OF DSN8CCGI.
      MOVE ZEROES TO AACTIONL OF DSN8CCGI.
      MOVE SPACES TO AACTIONI OF DSN8CCGI.
      MOVE ZEROES TO ADESCL2L OF DSN8CCGI.
      MOVE SPACES TO ADESCL2I OF DSN8CCGI.
      MOVE ZEROES TO AOBJECTL OF DSN8CCGI.
      MOVE SPACES TO AOBJECTI OF DSN8CCGI.
      MOVE ZEROES TO ADESCL3L OF DSN8CCGI.
      MOVE SPACES TO ADESCL3I OF DSN8CCGI.
      MOVE ZEROES TO ASEARCHL OF DSN8CCGI.
      MOVE SPACES TO ASEARCHI OF DSN8CCGI.
      MOVE ZEROES TO ADESCL4L OF DSN8CCGI.
      MOVE SPACES TO ADESCL4I OF DSN8CCGI.
      MOVE ZEROES TO ADATAL   OF DSN8CCGI.
      MOVE SPACES TO ADATAI   OF DSN8CCGI.
      MOVE ZEROES TO AMSGL    OF DSN8CCGI.
      MOVE SPACES TO AMSGI    OF DSN8CCGI.
      MOVE ZEROES TO ALINEL(1).
      MOVE SPACES TO ALINEI(1).
      MOVE ZEROES TO ALINEL(2).
      MOVE SPACES TO ALINEI(2).
      MOVE ZEROES TO ALINEL(3).
      MOVE SPACES TO ALINEI(3).
      MOVE ZEROES TO ALINEL(4).
      MOVE SPACES TO ALINEI(4).
      MOVE ZEROES TO ALINEL(5).
      MOVE SPACES TO ALINEI(5).
      MOVE ZEROES TO ALINEL(6).
      MOVE SPACES TO ALINEI(6).
      MOVE ZEROES TO ALINEL(7).
      MOVE SPACES TO ALINEI(7).
      MOVE ZEROES TO ALINEL(8).
      MOVE SPACES TO ALINEI(8).
      MOVE ZEROES TO ALINEL(9).
      MOVE SPACES TO ALINEI(9).
      MOVE ZEROES TO ALINEL(10).
      MOVE SPACES TO ALINEI(10).
      MOVE ZEROES TO ALINEL(11).
      MOVE SPACES TO ALINEI(11).
      MOVE ZEROES TO ALINEL(12).
      MOVE SPACES TO ALINEI(12).
      MOVE ZEROES TO ALINEL(13).
      MOVE SPACES TO ALINEI(13).
      MOVE ZEROES TO ALINEL(14).
      MOVE SPACES TO ALINEI(14).
      MOVE ZEROES TO ALINEL(15).
      MOVE SPACES TO ALINEI(15).
      MOVE ZEROES TO APFKEYL  OF DSN8CCGI.
      MOVE SPACES TO APFKEYI  OF DSN8CCGI.
 * INIT AREA INCLUDED BY DSN8MCMD                              @01
      MOVE ZEROES TO BTITLEL  OF DSN8CCDI.
      MOVE SPACES TO BTITLEI  OF DSN8CCDI.
      MOVE ZEROES TO BMAJSYSL OF DSN8CCDI.
      MOVE SPACES TO BMAJSYSI OF DSN8CCDI.
      MOVE ZEROES TO BACTIONL OF DSN8CCDI.
      MOVE SPACES TO BACTIONI OF DSN8CCDI.
      MOVE ZEROES TO BDESCL2L OF DSN8CCDI.
      MOVE SPACES TO BDESCL2I OF DSN8CCDI.
      MOVE ZEROES TO BOBJECTL OF DSN8CCDI.
```

```
           MOVE SPACES TO BOBJECTI OF DSN8CCDI.
           MOVE ZEROES TO BDESCL3L OF DSN8CCDI.
           MOVE SPACES TO BDESCL3I OF DSN8CCDI.
           MOVE ZEROES TO BSEARCHL OF DSN8CCDI.
           MOVE SPACES TO BSEARCHI OF DSN8CCDI.
           MOVE ZEROES TO BDESCL4L OF DSN8CCDI.
           MOVE SPACES TO BDESCL4I OF DSN8CCDI.
           MOVE ZEROES TO BDATAL   OF DSN8CCDI.
           MOVE SPACES TO BDATAI   OF DSN8CCDI.
           MOVE ZEROES TO BMSGL    OF DSN8CCDI.
           MOVE SPACES TO BMSGI    OF DSN8CCDI.
           MOVE ZEROES TO LINE1F1L OF DSN8CCDI.
           MOVE SPACES TO LINE1F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE1F2L OF DSN8CCDI.
           MOVE SPACES TO LINE1F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE2F1L OF DSN8CCDI.
           MOVE SPACES TO LINE2F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE2F2L OF DSN8CCDI.
           MOVE SPACES TO LINE2F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE3F1L OF DSN8CCDI.
           MOVE SPACES TO LINE3F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE3F2L OF DSN8CCDI.
           MOVE SPACES TO LINE3F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE4F1L OF DSN8CCDI.
           MOVE SPACES TO LINE4F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE4F2L OF DSN8CCDI.
           MOVE SPACES TO LINE4F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE5F1L OF DSN8CCDI.
           MOVE SPACES TO LINE5F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE5F2L OF DSN8CCDI.
           MOVE SPACES TO LINE5F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE6F1L OF DSN8CCDI.
           MOVE SPACES TO LINE6F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE6F2L OF DSN8CCDI.
           MOVE SPACES TO LINE6F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE7F1L OF DSN8CCDI.
           MOVE SPACES TO LINE7F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE7F2L OF DSN8CCDI.
           MOVE SPACES TO LINE7F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE8F1L OF DSN8CCDI.
           MOVE SPACES TO LINE8F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE8F2L OF DSN8CCDI.
           MOVE SPACES TO LINE8F2I OF DSN8CCDI.
           MOVE ZEROES TO LINE9F1L OF DSN8CCDI.
           MOVE SPACES TO LINE9F1I OF DSN8CCDI.
           MOVE ZEROES TO LINE9F2L OF DSN8CCDI.
           MOVE SPACES TO LINE9F2I OF DSN8CCDI.
           MOVE ZEROES TO LINEAF1L OF DSN8CCDI.
           MOVE SPACES TO LINEAF1I OF DSN8CCDI.
           MOVE ZEROES TO LINEAF2L OF DSN8CCDI.
           MOVE SPACES TO LINEAF2I OF DSN8CCDI.
           MOVE ZEROES TO LINEBF1L OF DSN8CCDI.
           MOVE SPACES TO LINEBF1I OF DSN8CCDI.
           MOVE ZEROES TO LINEBF2L OF DSN8CCDI.
           MOVE SPACES TO LINEBF2I OF DSN8CCDI.
           MOVE ZEROES TO LINECF1L OF DSN8CCDI.
           MOVE SPACES TO LINECF1I OF DSN8CCDI.
           MOVE ZEROES TO LINECF2L OF DSN8CCDI.
           MOVE SPACES TO LINECF2I OF DSN8CCDI.
           MOVE ZEROES TO LINEDF1L OF DSN8CCDI.
           MOVE SPACES TO LINEDF1I OF DSN8CCDI.
           MOVE ZEROES TO LINEDF2L OF DSN8CCDI.
           MOVE SPACES TO LINEDF2I OF DSN8CCDI.
           MOVE ZEROES TO LINEEF1L OF DSN8CCDI.
           MOVE SPACES TO LINEEF1I OF DSN8CCDI.
           MOVE ZEROES TO LINEEF2L OF DSN8CCDI.
           MOVE SPACES TO LINEEF2I OF DSN8CCDI.
           MOVE ZEROES TO LINEFF1L OF DSN8CCDI.
           MOVE SPACES TO LINEFF1I OF DSN8CCDI.
           MOVE ZEROES TO LINEFF2L OF DSN8CCDI.
           MOVE SPACES TO LINEFF2I OF DSN8CCDI.
           MOVE ZEROES TO BPFKEYL  OF DSN8CCDI.
           MOVE SPACES TO BPFKEYI  OF DSN8CCDI.
      *
           MOVE 'DSN8CC0' TO MAJOR IN DSN8-MODULE-NAME.
           MOVE 'O'       TO MAJSYS IN OUTAREA.
           MOVE '0'       TO EXITCODE.
           MOVE EIBTRMID TO CICSID OF PCONVSTA.
           MOVE CONVID OF PCONVSTA TO SAVE-CONVID.

      *****************************************************************-
      *    TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE
```

```
*    LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL.
*****************************************************************-

     EXEC SQL SELECT LASTSCR
              INTO   :PCONA.LASTSCR
              FROM   VCONA
              WHERE  CONVID = :SAVE-CONVID END-EXEC.

*****************************************************************-
*    IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO
*    RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES
*    TO GET TITLE ETC. FOR OUTPUT MAP.
*****************************************************************-

     IF SQLCODE = +100 THEN
              GO TO CC0SEND.

*****************************************************************-
*    IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA  N
*    FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED N
*    SCREEN AND PRESSED ENTER THEN .......                       N
*    MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO
*    VALIDATION MODULES.
*****************************************************************-
     EXEC CICS HANDLE CONDITION MAPFAIL (CC0SEND) END-EXEC.

     IF LASTSCR OF PCONA  NOT = 'DSN8002 ' THEN
       GO TO CC0-LABEL1.

*                                              **DSN8002
*                                              **DETAIL MAP
*                                              **MOVE DATA INTO
*                                              **INPUT FIELDS

       EXEC CICS RECEIVE MAP ('DSN8CCD') MAPSET ('DSN8CCD')
            END-EXEC.
       IF BMAJSYSL NOT = 0 THEN MOVE BMAJSYSI TO MAJSYS OF INAREA
                         ELSE MOVE 'O'      TO MAJSYS OF INAREA.
       IF BACTIONL NOT = 0 THEN MOVE BACTIONI TO ACTION OF INAREA
                         ELSE MOVE SPACES   TO ACTION OF INAREA.
       IF BOBJECTL NOT = 0 THEN MOVE BOBJECTI TO OBJFLD OF INAREA
                         ELSE MOVE SPACES   TO OBJFLD OF INAREA.
       IF BSEARCHL NOT = 0 THEN MOVE BSEARCHI TO SRCH   OF INAREA
                         ELSE MOVE SPACES   TO SRCH   OF INAREA.
       IF BDATAL   NOT = 0 THEN MOVE BDATAI   TO DATAIN OF INAREA
                         ELSE MOVE SPACES   TO DATAIN OF INAREA.
       MOVE 1 TO I.

*                                       **GO TO VALIDATION MODULES

       GO TO CC0-LABELX.

*                                           **ERROR ON LASTSCREEN?

 CC0-LABEL1.
     IF LASTSCR OF PCONA NOT = 'DSN8001 ' THEN
       EXEC CICS ABEND ABCODE ('LSCR') END-EXEC
       GOBACK.

*                                           **USING LAST SCREEN
*                                           **SPECIFIED TO RECEIVE
*                                           **INPUT FROM TERMINAL

     EXEC CICS RECEIVE MAP ('DSN8CCG') MAPSET('DSN8CCG') END-EXEC.

*****************************************************************-
*    IF DATA IS RECEIVED FOR A FIELD, THEN MOVE THE DATA INTO THE
*    CORRESPONDING FIELD IN INAREA, OTHERWISE MOVE BLANKS.
*****************************************************************-

     IF AMAJSYSL NOT = 0 THEN MOVE AMAJSYSI TO MAJSYS OF INAREA
                         ELSE MOVE 'O'      TO MAJSYS OF INAREA.
     IF AACTIONL NOT = 0 THEN MOVE AACTIONI TO ACTION OF INAREA
                         ELSE MOVE SPACES   TO ACTION OF INAREA.
     IF AOBJECTL NOT = 0 THEN MOVE AOBJECTI TO OBJFLD OF INAREA
                         ELSE MOVE SPACES   TO OBJFLD OF INAREA.
     IF ASEARCHL NOT = 0 THEN MOVE ASEARCHI TO SRCH   OF INAREA
                         ELSE MOVE SPACES   TO SRCH   OF INAREA.
     IF ADATAL   NOT = 0 THEN MOVE ADATAI   TO DATAIN OF INAREA
                         ELSE MOVE SPACES   TO DATAIN OF INAREA.
     GO TO CC0-LABEL3.
```

```
 CC0-LABELX.
       IF COL2LEN(I) NOT = 0 THEN MOVE COL2DATA(I) TO TRANDATA(I)
                            ELSE MOVE SPACES   TO TRANDATA(I).
       ADD 1 TO I.

*                                  ** CC0-LABELX LOOP
 CC0-LOOPX.
       PERFORM CC0-LABELX UNTIL I > 15.

 CC0-LABEL3.
       MOVE 1 TO I.
       MOVE 0 TO FOUND.

*****************************************************************-
*    CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED
*    BY DSN8CC1 AND DSN8CC2 EG. PF1 = '01' AND PF13 = '01'.
*****************************************************************-
 CC0-LABEL4.

*                                         **PF KEYS 1-12

     IF PFKEYS(I) = EIBAID THEN MOVE 1 TO FOUND
         ELSE ADD 1 TO I.

*                                         ** CC0-LABEL4 LOOP
 CC0-LOOP4.
     PERFORM CC0-LABEL4 UNTIL
       I > 24 OR FOUND = 1.

*                                         **PF KEYS > 12
 CC0-LABEL5.
     IF I >  12 THEN SUBTRACT 12 FROM I.
     IF FOUND = 1 THEN
       MOVE PFK(I) TO PFKIN OF INAREA
     ELSE MOVE SPACES TO PFKIN OF INAREA.
     GO TO CC0-LABEL6.

**************************************************************** -
*    GO TO DSN8CC1, GET DCLGEN STRUCTURES AND TABLE DCL
*****************************************************************-

 CC0SEND.
     MOVE SPACES TO INAREA.
     MOVE '00' TO PFKIN OF INAREA.

 CC0-LABEL6.
     MOVE 'O' TO MAJSYS IN INAREA.
     EXEC CICS LINK PROGRAM ('DSN8CC1') COMMAREA(COMMAREA)
         LENGTH(3000) END-EXEC.
     GO TO CC0-NORMAL.
     EXEC SQL INCLUDE DSN8MCXX END-EXEC.

**************************************************************** -
*    AFTER RETURN FROM DSN8CC1, MOVE DATA TO OUTPUT MAP AREA AND
*    SEND MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVSTA.
*****************************************************************-
 CC0-NORMAL.
     IF LASTSCR OF PCONVSTA = 'DSN8002 ' THEN GO TO CC0-LABEL9.

*                                        **MOVE DATA INTO
*                                        **OUTPUT FIELDS
     MOVE HTITLE OF OUTAREA  TO ATITLEO.
     MOVE MAJSYS OF OUTAREA TO AMAJSYSO.
     MOVE ACTION OF OUTAREA TO AACTIONO.
     MOVE OBJFLD OF OUTAREA TO AOBJECTO.
     MOVE SRCH   OF OUTAREA TO ASEARCHO.
     MOVE DATAOUT           TO ADATAO.
     MOVE MSG    OF OUTAREA TO AMSGO.
     MOVE DESC2  OF OUTAREA TO ADESCL2O.
     MOVE DESC3  OF OUTAREA TO ADESCL3O.
     MOVE DESC4  OF OUTAREA TO ADESCL4O.
     MOVE PFKTEXT OF OUTAREA TO APFKEYO.
     MOVE 1 TO I.

*                                       **SEND MAP ACCORDING TO
*                                       **PREVIOUS SCREEN
 CC0-LABEL7.
     MOVE LINE0(I) TO ALINEO(I).
     ADD 1 TO I.

*                                       **CC0-LABEL7 LOOP
 CC0-LOOP7.
```

```
     PERFORM CC0-LABEL7 UNTIL
         I > 15.

*****************************************************************-
*    CREATES A DYNAMIC CURSOR
*****************************************************************-

*                                         **SET CURSOR POSITION
 CC0-LABEL8.
     MOVE ZEROES TO CURSOR-VALUE.
     IF AACTIONO = SPACES THEN MOVE +179 TO CURSOR-VALUE
     ELSE IF AOBJECTO = SPACES THEN MOVE +259 TO CURSOR-VALUE
     ELSE IF ASEARCHO = SPACES THEN MOVE +339 TO CURSOR-VALUE
     ELSE IF ADATAO = SPACES OR AACTIONO = 'D' OR 'E' THEN
     MOVE +419 TO CURSOR-VALUE.

*                                               **SEND OUTPUT MAP

     IF CURSOR-VALUE = ZEROES THEN
       EXEC CICS SEND MAP('DSN8CCG') MAPSET('DSN8CCG') END-EXEC
     ELSE
       EXEC CICS SEND MAP('DSN8CCG') MAPSET('DSN8CCG') ERASE
       CURSOR(CURSOR-VALUE) END-EXEC.

*                                                **FINISHED?

     IF EXITCODE = '1' THEN GO TO CC0-LABEL12.
     EXEC CICS RETURN TRANSID('D8CS') END-EXEC.

*****************************************************************-
*    MOVES DATA FROM OUTPUT MAP AREA TO
*    RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST
*****************************************************************-

*                                        **MOVE DATA
*                                        **FROM OUTPUT FIELDS
 CC0-LABEL9.
     MOVE HTITLE OF OUTAREA TO BTITLEO.
     MOVE MAJSYS OF OUTAREA TO BMAJSYSO.
     MOVE ACTION OF OUTAREA TO BACTIONO.
     MOVE OBJFLD OF OUTAREA TO BOBJECTO.
     MOVE SRCH   OF OUTAREA TO BSEARCHO.
     MOVE DATAOUT           TO BDATAO.
     MOVE MSG    OF OUTAREA TO BMSGO.
     MOVE DESC2  OF OUTAREA TO BDESCL2O.
     MOVE DESC3  OF OUTAREA TO BDESCL3O.
     MOVE DESC4  OF OUTAREA TO BDESCL4O.
     MOVE PFKTEXT OF OUTAREA TO BPFKEYO.
     MOVE 1 TO I.
*                                        **RECEIVE MAP ACCORDING
*                                           **TO PREVIOUS SCREEN

 CC0-LABEL10.
     MOVE FIELD1(I)  TO COL1DATA(I) .
*                          ** CHECK FOR ATTRIBUTE OF X'C0C1'
     IF ATTR(I) = -16191 THEN MOVE -1 TO COL2LEN(I).
     MOVE ATTR2(I)   TO COL2ATTR(I) .
     MOVE FIELD2(I)  TO COL2DATA(I) .
     ADD 1 TO I.

*                          ** CC0-LABEL10 LOOP
 CC0-LOOP10.
     PERFORM CC0-LABEL10 UNTIL
        I > 15.

 CC0-LABEL11.
*****************************************************************-
*    CREATES A DYNAMIC CURSOR
*****************************************************************-

*                                         **SET CURSOR POSITION
     MOVE ZEROES TO CURSOR-VALUE.
     IF BACTIONO = SPACES THEN MOVE +179 TO CURSOR-VALUE
     ELSE IF BOBJECTO = SPACES THEN MOVE +259 TO CURSOR-VALUE
     ELSE IF BSEARCHO = SPACES THEN MOVE +339 TO CURSOR-VALUE
     ELSE IF BDATAO = SPACES OR BACTIONO = 'D' OR 'E' THEN
     MOVE +419 TO CURSOR-VALUE.

*                                              **SEND INPUT MAP

     IF CURSOR-VALUE = ZEROES THEN
         EXEC CICS SEND MAP('DSN8CCD') MAPSET('DSN8CCD') END-EXEC
```

```
     ELSE
         EXEC CICS SEND MAP('DSN8CCD') MAPSET('DSN8CCD') ERASE
         CURSOR(CURSOR-VALUE) END-EXEC.

*                                               **FINISHED?
     IF EXITCODE = '1' THEN GO TO CC0-LABEL12.
     EXEC CICS RETURN TRANSID('D8CS') END-EXEC.
     GOBACK.


*                                              **RETURN
 CC0-LABEL12.
     EXEC CICS RETURN END-EXEC.
     GOBACK.
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CC1

THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS THE PARAMETER AREA.

```
 IDENTIFICATION DIVISION.
*------------------------
 PROGRAM-ID. DSN8CC1.

********** DSN8CC1 - SQL 1 MAINLINE FOR CICS - COBOL ************
*                                                               *
* MODULE NAME = DSN8CC1                                         *
*                                                               *
* DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                    *
*                    SQL 1 MAINLINE                             *
*                    CICS                                       *
*                    COBOL                                      *
*                                                               *
*Licensed Materials - Property of IBM                           *
*5605-DB2                                                       *
*(C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.       *
*                                                               *
*STATUS = Version 10                                            *
*                                                               *
* FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE  *
*            SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS    *
*            THE PARAMETER AREA.                                *
*                                                               *
* NOTES =                                                       *
*   DEPENDENCIES = CALLED BY DSN8CC0, CALLS DSN8CC2(CICS LINKS).*
*   RESTRICTIONS = NONE                                         *
*                                                               *
*                                                               *
* MODULE TYPE =                                                 *
*    PROCESSOR   = DB2 PRECOMPILER,CICS TRANSLATOR,COBOL COMPILER
*    MODULE SIZE = SEE LINK_EDIT                                *
*    ATTRIBUTES  = REUSABLE                                     *
*                                                               *
* ENTRY POINT =  DSN8CC1                                        *
*    PURPOSE = SEE FUNCTION                                     *
*    LINKAGE = INCLUDED BY MODULE DSN8MC1                       *
*                                                               *
*    INPUT  = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:    *
*                                                               *
*            SYMBOLIC LABEL/NAME = NONE                         *
*            DESCRIPTION = NONE                                 *
*                                                               *
*    OUTPUT  = PARAMETERS EXPLICITLY RETURNED:                  *
*                                                               *
*             SYMBOLIC LABEL/NAME = NONE                        *
*             DESCRIPTION = NONE                                *
*                                                               *
* EXIT-NORMAL = DSN8CC0                                         *
*                                                               *
* EXIT-ERROR = DSN8CC0                                          *
*                                                               *
*     RETURN CODE = NONE                                        *
*                                                               *
```

```
*     ABEND CODES = NONE                                        *
*                                                               *
*     ERROR-MESSAGES =  NONE                                    *
*                                                               *
* EXTERNAL REFERENCES =                                         *
*   ROUTINES/SERVICES =  DSN8CC2                                *
*                                                               *
*   DATA-AREAS =                                                *
*       DSN8MCCA          - COBOL STRUCTURE FOR DFHCOMMAREA     *
*       DSN8MCCS          - VCONA TABLE DCL AND PCONA DCLGEN    *
*       DSN8MCC2          - COMMON AREA PART 2                  *
*       DSN8MCOV          - VOPTVAL TABLE DCL & POPTVAL DCLGEN  *
*       DSN8MCVO          - FINDS VALID OPTIONS FOR ACTION,     *
*                           OBJECT, SEARCH CRITERIA             *
*       DSN8MC1           - SQL1 COMMON MODULE FOR IMS AND CICS *
*       DSN8MC3 - DSN8MC5 - VALIDATION MODULES CALLED BY DSN8MC0*
*       DSN8MCXX          - SQL ERROR HANDLER                   *
*                                                               *
*   CONTROL-BLOCKS =                                            *
*       SQLCA             - SQL COMMUNICATION AREA              *
*                                                               *
* TABLES = NONE                                                 *
*                                                               *
* CHANGE-ACTIVITY =                                             *
* - 10/18/2005  PK03311  INITIALIZE UNINITIALIZED STORAGE   @01 *
*                                                               *
*  *PSEUDOCODE*                                                 *
*                                                               *
*     PROCEDURE                                                 *
*      INCLUDE DECLARATIONS.                                    *
*      INCLUDE DSN8MC1.                                         *
*      INCLUDE ERROR HANDLER.                                   *
*                                                               *
*      CC1EXIT: ( REFERENCED BY DSN8MC1 )                       *
*        EXEC CICS RETURN.                                      *
*                                                               *
*      CC1CALL: ( REFERENCED BY DSN8MC1 )                       *
*          EXEC CICS LINK PROGRAM('DSN8CC2')                    *
*                              COMMAREA(DFHCOMMAREA).           *
*         GO TO MC1SAVE.  (LABEL IN DSN8MC1)                    *
*                                                               *
*      INCLUDE VALIDATION MODULES.                              *
*                                                               *
*     END.                                                      *
*****************************************************************
 ENVIRONMENT DIVISION.
*------------------------

 DATA DIVISION.
*------------------------
 WORKING-STORAGE SECTION.
*****************************************************************
*      * DECLARE FIELD PASSED TO MESSAGE ROUTINE                *
*      * DECLARE CONVERSATION STATUS                            *
*      * DECLARE MESSAGE TEXT                                   *
*      * DECLARE OPTION VALIDATION                              *
*      * DECLARE COMMON AREA AND COMMON AREA PART 2             *
*****************************************************************
  01 MSGCODE                PIC X(04).

  01 OUTMSG                 PIC X(69).

     EXEC SQL INCLUDE DSN8MCCS END-EXEC.
     EXEC SQL INCLUDE DSN8MCOV END-EXEC.
     EXEC SQL INCLUDE SQLCA    END-EXEC.
     EXEC SQL INCLUDE DSN8MCC2 END-EXEC.

 LINKAGE SECTION.
 01  DFHCOMMAREA.
     EXEC SQL INCLUDE DSN8MCCA END-EXEC.

 PROCEDURE DIVISION.
*------------------------

* INIT AREA INCLUDED BY DSN8MCCS                              @01
     MOVE SPACES TO PCONA.
* INIT AREA INCLUDED BY DSN8MCOV                              @01
     MOVE SPACES TO POPTVAL.

********************************************************
*  SQL RETURN CODE HANDLING                            *
********************************************************
```

```
     EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
     EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.


     MOVE 'DSN8CC1' TO MAJOR IN DSN8-MODULE-NAME.

************************************************************
*  FIND VALID OPTIONS FOR ACTION, OBJECT, SEARCH CRITERION*
*  RETRIEVE CONVERSATION, VALIDATE, CALL SQL2            *
************************************************************

     EXEC SQL INCLUDE DSN8MCVO END-EXEC.
     EXEC SQL INCLUDE DSN8MC1 END-EXEC.

*                                          **RETURN
 CC1-EXIT.
     EXEC CICS RETURN END-EXEC.

************************************************************
*  VALIDATE ACTION, OBJECT, SEARCH CRITERIA              *
*  HANDLE ERRORS                                         *
************************************************************

 CC1-CALL.
     EXEC CICS LINK PROGRAM('DSN8CC2') COMMAREA(DFHCOMMAREA)
              LENGTH(3000) END-EXEC.
     GO TO MC1-SAVE.
     EXEC SQL INCLUDE DSN8MC3 END-EXEC.
     EXEC SQL INCLUDE DSN8MC4 END-EXEC.
     EXEC SQL INCLUDE DSN8MC5 END-EXEC.
     EXEC SQL INCLUDE DSN8MCXX END-EXEC.
     GOBACK.
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CC2

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING CALLS SECONDARY SELECTION MODULES DSN8MCA DSN8MCM CALLS DETAIL MODULES DSN8MCD DSN8MCE DSN8MCF DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ CALLED BY DSN8MC1 (SQL1) .

```
 IDENTIFICATION DIVISION.
*------------------------
 PROGRAM-ID. DSN8CC2.

************* DSN8CC2 - COMMON MODULE FOR CICS - COBOL***********
*                                                               *
*   MODULE NAME = DSN8CC2                                       *
*                                                               *
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                  *
*                      SQL 2 COMMON MODULE                      *
*                      CICS                                     *
*                      COBOL                                    *
*                                                               *
*Licensed Materials - Property of IBM                           *
*5605-DB2                                                       *
*(C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.       *
*                                                               *
*STATUS = Version 10                                            *
*                                                               *
*                                                               *
*                                                               *
*   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR            *
*                         DETAIL PROCESSING                     *
*              CALLS SECONDARY SELECTION MODULES                *
*                    DSN8MCA DSN8MCM                            *
*              CALLS DETAIL MODULES                             *
*                    DSN8MCD DSN8MCE DSN8MCF                    *
*                    DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ    *
*              CALLED BY DSN8MC1 (SQL1)                         *
*                                                               *
*   NOTES =  NONE                                               *
*                                                               *
*                                                               *
```

```
*   MODULE TYPE =                                                 *
*      PROCESSOR   = DB2 PRECOMPILER, CICS TRANSLATOR,            *
*                    VS COBOL                                     *
*      MODULE SIZE = SEE LINKEDIT                                 *
*      ATTRIBUTES  = REUSABLE                                     *
*                                                                 *
*   ENTRY POINT = DSN8CC2                                         *
*      PURPOSE = SEE FUNCTION                                     *
*      LINKAGE = NONE                                             *
*      INPUT   =                                                  *
*                                                                 *
*                SYMBOLIC LABEL/NAME = COMMPTR                    *
*                DESCRIPTION         = POINTER TO COMMAREA        *
*                                      (COMMUNICATION AREA)       *
*                                                                 *
*      OUTPUT  =                                                  *
*                                                                 *
*                SYMBOLIC LABEL/NAME = COMMPTR                    *
*                DESCRIPTION         = POINTER TO COMMAREA        *
*                                      (COMMUNICATION AREA)       *
*                                                                 *
*   EXIT-NORMAL =  RETURN CODE 0 NORMAL COMPLETION                *
*                                                                 *
*   EXIT-ERROR =                                                  *
*            IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION  *
*            SPECIFIED IN DSN8CC2 WILL BE RAISED AND PROGRAM    *
*            WILL GO TO THE LABEL DB-ERROR.                       *
*                                                                 *
*                                                                 *
*      RETURN CODE = NONE                                         *
*                                                                 *
*                                                                 *
*      ABEND CODES = NONE                                         *
*                                                                 *
*      ERROR-MESSAGES =                                           *
*         DSN8062E-AN OBJECT WAS NOT SELECTED                     *
*         DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR                 *
*         DSN8072E-INVALID SELECTION ON SECONDARY SCREEN          *
*                                                                 *
*   EXTERNAL REFERENCES = NONE                                    *
*      ROUTINES/SERVICES = 10 MODULES LISTED ABOVE                *
*         DSN8MCG    -   ERROR MESSAGE ROUTINE                    *
*                                                                 *
*      DATA-AREAS =                                               *
*         DSN8MCA    -   SECONDARY SELECTION FOR ORGANIZATION   *
*         DSN8MCAD   -   DECLARE ADMINISTRATION DETAIL            *
*         DSN8MCAE   -   CURSOR EMPLOYEE LIST                     *
*         DSN8MCAL   -   CURSOR ADMINISTRATION LIST               *
*         DSN8MCA2   -   DECLARE ADMINISTRATION DETAIL            *
*         DSN8MCCA   -   COMMON AREA                              *
*         DSN8MCC2   -   COMMON AREA PART 2                       *
*         DSN8MCD    -   DEPARTMENT STRUCTURE DETAIL              *
*         DSN8MCDA   -   CURSOR ADMINISTRATION DETAIL             *
*         DSN8MCDH   -   CURSOR FOR DISPLAY TEXT FROM             *
*                        TDSPTXT TABLE                            *
*         DSN8MCDM   -   DECLARE DEPARTMENT MANAGER               *
*         DSN8MCDP   -   DECLARE DEPARTMENT                       *
*         DSN8MCDT   -   DECLARE DISPLAY TEXT                     *
*         DSN8MCE    -   DEPARTMENT DETAIL                        *
*         DSN8MCEM   -   DECLARE EMPLOYEE                         *
*         DSN8MCED   -   DECLARE EMPLOYEE-DEPARTMENT              *
*         DSN8MCF    -   EMPLOYEE   DETAIL                        *
*         DSN8MCOV   -   DECLARE OPTION VALIDATION                *
*         DSN8MCXX   -   ERROR HANDLER                            *
*                                                                 *
*      CONTROL-BLOCKS =                                           *
*         SQLCA      -   SQL COMMUNICATION AREA                   *
*                                                                 *
*   TABLES = NONE                                                 *
*                                                                 *
*   CHANGE-ACTIVITY =                                             *
*   - ADD NEW VARIABLES FOR REFERENTIAL INTEGRITY          V2R1 *
*   - 10/18/2005  PK03311  INITIALIZE UNINITIALIZED STORAGE @01 *
*                                                                 *
*  *PSEUDOCODE*                                                   *
*                                                                 *
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR        *
* DETAIL MODULE(S) ARE TO BE CALLED IN THE CICS/COBOL            *
* ENVIRONMENT.                                                    *
*                                                                 *
* WHAT HAS HAPPENED SO FAR?.............THE SUBSYSTEM            *
* DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS                 *
```

```
      * READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL *
      * TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT   *
      * FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF    *
      * ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS      *
      * MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH     *
      * POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE    *
      * BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION      *
      * AND DETAIL MODULES.                                           *
      *                                                               *
      *WHAT IS INCLUDED IN THIS MODULE?.............                  *
      * ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.    *
      * ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE        *
      * SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND                *
      * SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE     *
      * RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE*
      * THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED  *
      * IN THIS PROCEDURE.                                            *
      *                                                               *
      * PROCEDURE                                                     *
      *    IF ANSWER TO DETAIL SCREEN AND DETAIL PROCESSOR            *
      *    IS NOT WILLING TO ACCEPT AN ANSWER THEN                    *
      *                    NEW REQUEST*                               *
      *                                                               *
      *    ELSE                                                       *
      *           IF ANSWER TO A SECONDARY SELECTION THEN             *
      *                    DETERMINE IF NEW REQUEST.                  *
      *                                                               *
      *                                                               *
      *                                                               *
      *    CASE (NEW REQUEST)                                         *
      *                                                               *
      *           SUBCASE ('ADD')                                     *
      *                   DETAIL PROCESSOR                            *
      *                   RETURN TO SQL 1                             *
      *           ENDSUB                                              *
      *                                                               *
      *           SUBCASE ('ERASE','DISPLAY','UPDATE')                *
      *                   CALL SECONDARY SELECTION                    *
      *                   IF # OF POSSIBLE CHOICES IS ^= 1 THEN       *
      *                          RETURN TO SQL 1                      *
      *                   ELSE                                        *
      *                          CALL THE DETAIL PROCESSOR            *
      *                          RETURN TO SQL 1                      *
      *           ENDSUB                                              *
      *                                                               *
      *    ENDCASE                                                    *
      *                                                               *
      *    IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS       *
      *    ACTUALLY BEEN MADE THEN                                    *
      *                IF IT IS A VALID SELECTION NUMBER THEN         *
      *                      CALL DETAIL PROCESSOR                    *
      *                      RETURN TO SQL 1                          *
      *                END                                            *
      *                                                               *
      *                ELSE                                           *
      *                      PRINT ERROR MSG                          *
      *                      RETURN TO SQL 1                          *
      *                END.                                           *
      *                                                               *
      *    IF ANSWER TO SECONDARY SELECTION THEN                      *
      *             CALL SECONDARY SELECTION                          *
      *             RETURN TO SQL 1                                   *
      *    END.                                                       *
      *                                                               *
      *    IF ANSWER TO DETAIL THEN                                   *
      *             CALL DETAIL PROCESSOR                             *
      *             RETURN TO SQL 1                                   *
      *    END.                                                       *
      *                                                               *
      *    RETURN TO SQL 1.                                           *
      *                                                               *
      * END.                                                          *
      *                                                               *
      *   *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES*
      *    THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.        *
      *---------------------------------------------------------------*


       ENVIRONMENT DIVISION.
      *---------------------

       DATA DIVISION.
```

```
*-------------
 WORKING-STORAGE SECTION.

***************************************
*  FIELDS SENT TO MESSAGE ROUTINE     *
***************************************
  01 MSGCODE                PIC X(04).

  01 OUTMSG                 PIC X(69).

***************************************
*  NULL INDICATOR                     *
***************************************
  01 NULLIND1               PIC S9(4) COMP-4.
  01 NULLIND2               PIC S9(4) COMP-4.
  01 NULLIND3               PIC S9(4) COMP-4.
  01 NULLIND4               PIC S9(4) COMP-4.
  01 NULLIND5               PIC S9(4) COMP-4.
  01 NULLARRY.
     03  NULLARRY1  PIC S9(4) USAGE COMP OCCURS 13 TIMES.

     EXEC SQL INCLUDE SQLCA END-EXEC.

     EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
     EXEC SQL INCLUDE DSN8MCDP END-EXEC.
     EXEC SQL INCLUDE DSN8MCEM END-EXEC.
     EXEC SQL INCLUDE DSN8MCDM END-EXEC.
     EXEC SQL INCLUDE DSN8MCAD END-EXEC.
     EXEC SQL INCLUDE DSN8MCA2 END-EXEC.
     EXEC SQL INCLUDE DSN8MCOV END-EXEC.
     EXEC SQL INCLUDE DSN8MCDT END-EXEC.
     EXEC SQL INCLUDE DSN8MCED END-EXEC.

  01  CONSTRAINTS.
      03  PARM-LENGTH       PIC S9(4) COMP-4.
      03  REF-CONSTRAINT    PIC X(08).
      03  FILLER            PIC X(62).
  01  MGRNO-CONSTRAINT      PIC X(08)  VALUE 'RDE     '.

 LINKAGE SECTION.
 01  DFHCOMMAREA.
     EXEC SQL INCLUDE DSN8MCCA END-EXEC.


 PROCEDURE DIVISION.
*------------------
     EXEC SQL INCLUDE DSN8MCAE END-EXEC.
     EXEC SQL INCLUDE DSN8MCAL END-EXEC.
     EXEC SQL INCLUDE DSN8MCDH END-EXEC.
     EXEC SQL INCLUDE DSN8MCDA END-EXEC.

***********************************************************
*  SQL RETURN CODE HANDLING                               *
***********************************************************
     EXEC SQL WHENEVER SQLERROR   GO TO DB-ERROR END-EXEC
     EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

***********************************************************
*  INITIALIZATIONS                                        *
***********************************************************
     MOVE 'DSN8CC2' TO MAJOR.
     MOVE  SPACES   TO MINOR.
* INIT AREA INCLUDED BY DSN8MCDP                             @01
     MOVE SPACES TO PDEPT.
* INIT AREA INCLUDED BY DSN8MCEM                             @01
     MOVE SPACES TO PEMP.
* INIT AREA INCLUDED BY DSN8MCDM                             @01
     MOVE SPACES TO PDEPMGR.
* INIT AREA INCLUDED BY DSN8MCAD                             @01
     MOVE SPACES TO PASTRDET.
* INIT AREA INCLUDED BY DSN8MCA2                             @01
     MOVE SPACES TO PASTRDE2.
* INIT AREA INCLUDED BY DSN8MCOV                             @01
     MOVE SPACES TO POPTVAL.
* INIT AREA INCLUDED BY DSN8MCDT                             @01
     MOVE SPACES TO PDSPTXT.
* INIT AREA INCLUDED BY DSN8MCED                             @01
     MOVE SPACES TO PEMPDPT1.

***********************************************************
*  DETERMINES WHETHER NEW REQUEST OR NOT                  *
***********************************************************
```

```
 IC200B.

     IF PREV OF PCONVSTA = ' ' THEN
        MOVE 'Y' TO NEWREQ OF COMPARM.

     IF NEWREQ OF COMPARM = 'N' AND PREV OF PCONVSTA = 'S'
        AND DATA01 NOT = ' '
        AND PFKIN  NOT = '08'
        THEN MOVE 'Y' TO NEWREQ OF COMPARM.

     IF NEWREQ OF COMPARM NOT = 'Y' THEN
        GO TO IC2010.

************************************************************
*  IF NEW REQUEST AND ACTION IS 'ADD' THEN                 *
*       CALL DETAIL PROCESSOR                              *
*  ELSE CALL SECONDARY SELECTION                           *
************************************************************
     IF ACTION OF INAREA = 'A' THEN
*                                      **DETAIL PROCESSOR
        GO TO DETAIL0.
*                                      **SECONDARY SELECTION
     PERFORM SECSEL THRU END-SECSEL.
*                                       **IF NO. OF CHOICES = 1
*                                       **GO TO DETAIL PROCESSOR
     IF MAXSEL = 1 THEN
        GO TO DETAIL0.
     GO TO EXITO.
************************************************************
*  DETERMINES IF VALID SELECTION NUMBER                    *
************************************************************
 IC2010.
*                                     **VALID SELECTION NO. GIVEN
     IF PREV OF PCONVSTA NOT = 'S' OR
        MAXSEL < 1 OR
        PFKIN  = '08' OR
        DATA2 = DATO2 THEN
        GO TO IC201.

*                                        **DETAIL SELECTION GIVEN
     IF DAT1 NUMERIC AND DAT2 = ' ' THEN
        MOVE DAT1 TO DAT2
        MOVE '0' TO DAT1.
     IF DATA2 NUMERIC
        AND DATA2 > '00' AND DATA2 NOT > MAXSEL THEN
        MOVE 'Y' TO NEWREQ OF COMPARM
        GO TO DETAIL0.
*                                      **INVALID SELECTION NO.
*                                      **PRINT ERROR MESSAGE
     MOVE '072E' TO MSGCODE.
     CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
     MOVE OUTMSG TO MSG OF OUTAREA.
     GO TO EXITO.

************************************************************
*  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL        *
************************************************************
 IC201.
*                                           **SECONDARY SELECTION
     IF PREV OF PCONVSTA = 'S' THEN
        PERFORM SECSEL THRU END-SECSEL
        GO TO EXITO
     ELSE

*                                           **DETAIL PROCESSOR
        IF PREV OF PCONVSTA = 'D' THEN GO TO DETAIL0.

*                                           **LOGIC ERROR
*                                           **PRINT ERROR MESSAGE
     MOVE '066E' TO MSGCODE.
     CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
     MOVE OUTMSG TO MSG OF OUTAREA.
     GO TO EXITO.

*                                           **HANDLES ERRORS
     EXEC SQL INCLUDE DSN8MCXX END-EXEC.
     GO TO EXITO.

************************************************************
*  CALLS SECONDARY SELECTION AND RETURNS TO SQL 1          *
************************************************************
 SECSEL.
```

```
          MOVE 'DSN8001' TO LASTSCR IN PCONVSTA.
     *                                      **ADMINISTRATIVE
     *                                      **DEPARTMENT STRUCTURE
          IF OBJFLD OF INAREA = 'DS' THEN
             PERFORM DSN8MCA THRU END-DSN8MCA
          ELSE
     *                                     **INDIVIDUAL DEPARTMENT
     *                                     **PROCESSING
             IF OBJFLD OF INAREA = 'DE' THEN
                PERFORM DSN8MCA THRU END-DSN8MCA
             ELSE
     *                                     **INDIVIDUAL EMPLOYEE
     *                                     **PROCESSING
                IF OBJFLD OF INAREA = 'EM' THEN
                   PERFORM DSN8MCA THRU END-DSN8MCA
                ELSE

     *                                    **ERROR MESSAGE
     *                                    **MISSING SECONDARY SEL
                   MOVE '062E' TO MSGCODE
                   CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
                   MOVE OUTMSG TO MSG OF OUTAREA
                   GO TO EXITO.
      END-SECSEL.

     ***********************************************************
     *  CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1            *
     ***********************************************************
      DETAIL0.
          MOVE 'DSN8002' TO LASTSCR IN PCONVSTA.
     *                                     **ADMINISTRATIVE
     *                                     **DEPARTMENT STRUCTURE
          IF OBJFLD OF INAREA = 'DS' THEN
             PERFORM DSN8MCD THRU END-DSN8MCD
          ELSE
     *                                     **INDIVIDUAL DEPARTMENT
     *                                     **PROCESSING
             IF OBJFLD OF INAREA = 'DE' THEN
                PERFORM DSN8MCE THRU END-DSN8MCE
             ELSE
     *                                     **INDIVIDUAL EMPLOYEE
     *                                     **PROCESSING
                IF OBJFLD OF INAREA = 'EM' THEN
                   PERFORM DSN8MCF THRU END-DSN8MCF
                ELSE

     *                                    **ERROR MESSAGE
     *                                    **MISSING DETAIL MODULE
                   MOVE '062E' TO MSGCODE
                   CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
                   MOVE OUTMSG TO MSG OF OUTAREA.
          GO TO EXITO.

     *                                     **RETURNS TO SQL 1
      EXITO.
          EXEC CICS RETURN END-EXEC.

          EXEC SQL INCLUDE DSN8MCA END-EXEC.
          EXEC SQL INCLUDE DSN8MCD END-EXEC.
          EXEC SQL INCLUDE DSN8MCE END-EXEC.
          EXEC SQL INCLUDE DSN8MCF END-EXEC.
          GOBACK.
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CP0

THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE INPUT, CALLS DSN8CP1, AND ISSUES A CICS SEND MAP AFTER RETURNING.

```
DSN8CP0: PROC  OPTIONS (MAIN);                                         00010000
/********************************************************************* 00020000
*                                                                    * 00030000
*   MODULE NAME = DSN8CP0                                            * 00040000
*                                                                    * 00050000
```

```
 *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                        * 00060000
 *                      SUBSYSTEM INTERFACE MODULE                     * 00070000
 *                      CICS                                           * 00080000
 *                      PL/I                                           * 00090000
 *                      ORGANIZATION APPLICATION                       * 00100000
 *                                                                     * 00110000
 *     LICENSED MATERIALS - PROPERTY OF IBM 5655-DB2                   * 00120000
 *     (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.        * 00130000
 *                                                                     * 00140000
 *     STATUS = VERSION 10                                             * 00150000
 *                                                                     * 00160000
 *   FUNCTION = THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE      * 00170000
 *              INPUT, CALLS DSN8CP1, AND ISSUES A CICS SEND           * 00180000
 *               MAP AFTER RETURNING.                                  * 00190000
 *                                                                     * 00200000
 *   NOTES =                                                           * 00210000
 *         1.THIS IS A CICS PSEUDO CONVERSATION PROGRAM WHICH          * 00220000
 *           INITIALIZES ITSELF WHEN TERMINAL OPERATOR ENTERS          * 00230000
 *           INPUT AFTER VIEWING THE SCREEN SENT BY PREVIOUS           * 00240000
 *           ITERATIONS OF THE PROGRAM.                                * 00250000
 *                                                                     * 00260000
 *      DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED:             * 00270000
 *                              DSN8MCMG AND DSN8MCMD                  * 00280000
 *                     MODULES DSN8CP1 IS REQUIRED.                    * 00290000
 *                     DCLGEN STRUCTURE DSN8MPCS IS REQUIRED.          * 00300000
 *                     INCLUDED PLI STRUCTURE DSN8MPCA IS REQUIRED.    * 00310000
 *                                                                     * 00320000
 *      RESTRICTIONS = NONE                                            * 00330000
 *                                                                     * 00340000
 *                                                                     * 00350000
 *   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                             * 00360000
 *      PROCESSOR   = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZE * 00370000
 *      MODULE SIZE = SEE LINK-EDIT                                    * 00380000
 *      ATTRIBUTES  = REUSABLE                                         * 00390000
 *                                                                     * 00400000
 *   ENTRY POINT = DSN8CP0                                             * 00410000
 *      PURPOSE = SEE FUNCTION                                         * 00420000
 *      LINKAGE = CICS/OS/VS ENTRY                                     * 00430000
 *                                                                     * 00440000
 *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:         * 00450000
 *              SYMBOLIC LABEL/NAME =  DSN8CPDI                        * 00460000
 *              DESCRIPTION = CICS BMS MAP FOR DETAIL INPUT            * 00470000
 *                                                                     * 00480000
 *              SYMBOLIC LABEL/NAME =  DSN8CPGI                        * 00490000
 *              DESCRIPTION = CICS BMS MAP FOR GENERAL INPUT           * 00500000
 *                                                                     * 00510000
 *      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                       * 00520000
 *               SYMBOLIC LABEL/NAME =  DSN8CPDO                       * 00530000
 *               DESCRIPTION = CICS BMS MAP FOR DETAIL OUTPUT          * 00540000
 *                                                                     * 00550000
 *               SYMBOLIC LABEL/NAME =  DSN8CPGO                       * 00560000
 *               DESCRIPTION = CICS BMS MAP FOR GENERAL OUTPUT         * 00570000
 *                                                                     * 00580000
 *   EXIT-NORMAL = CICS RETURN TRANSID(D8PS).                          * 00590000
 *                                                                     * 00600000
 *   EXIT-ERROR = DB_ERROR FOR SQL ERRORS.                             * 00610000
 *                NO PL/I ON CONDITIONS.                               * 00620000
 *                                                                     * 00630000
 *      RETURN CODE = NONE                                             * 00640000
 *                                                                     * 00650000
 *      ABEND CODES =                                                  * 00660000
 *                CICS ABEND FOR CICS PROBLEMS.                        * 00670000
 *                MAPI - LASTSCREEN IS WRONG NAME ON INPUT             * 00680000
 *                MAPO - SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME * 00690000
 *                                                                     * 00700000
 *      ERROR-MESSAGES = NONE                                          * 00710000
 *                                                                     * 00720000
 *   EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS                    * 00730000
 *      ROUTINES/SERVICES = DSN8CP1                                    * 00740000
 *                                                                     * 00750000
 *      DATA-AREAS        =                                            * 00760000
 *         DSN8MPCA            - PARAMETER TO BE PASSED TO DSN8CP1     * 00770000
 *                               COMMON AREA                           * 00780000
 *         DSN8MPCS            - DECLARE CONVERSATION STATUS           * 00790000
 *         DSN8MPMD            - CICS/OS/VS PL/I MAP, ORGANIZATION     * 00800000
 *         DSN8MPMG            - CICS/OS/VS PL/I MAP, ORGANIZATION     * 00810000
 *                                                                     * 00820000
 *      CONTROL-BLOCKS    =                                            * 00830000
 *         SQLCA               - SQL COMMUNICATION AREA                * 00840000
 *                                                                     * 00850000
 *   TABLES = NONE                                                     * 00860000
 *                                                                     * 00870000
```

```
 *   CHANGE-ACTIVITY = NONE                                             *
 *                                                                      *
 *                                                                      *
 *  *PSEUDOCODE*                                                        *
 *                                                                      *
 *   PROCEDURE                                                          *
 *      DECLARATIONS.                                                   *
 *      ALLOCATE PLI WORK AREA FOR COMMAREA.                            *
 *         PUT MODULE NAME 'DSN8CP0' IN AREA USED BY ERROR-HANDLER.   *
 *         PUT CICS EIBTRMID IN PCONVSTA.CONVID TO BE PASSED TO DSN8CP1
 *         RETRIEVE LASTSCR FROM VCONA USING THE CONVID TO DETERMINE  *
 *         WHICH OF THE TWO BMS MAPS SHOULD BE USED TO MAP IN DATA.   *
 *                                                                      *
 *      IF RETRIEVAL IS SUCCESSFUL, THEN DO.                            *
 *          EXEC CICS RECEIVE MAP ACCORDING TO SPECIFIED LASTSCR.     *
 *          IF MAPFAIL CONDITION IS RAISED* THEN DO.                    *
 *                  COMPARM.PFKIN = '00'                                *
 *                  GO TO CP0CP1                                        *
 *           END                                                        *
 *                                                                      *
 *          ELSE                                                        *
 *                  PUT DATA FROM MAP INTO COMPARM **                   *
 *      ELSE                                                            *
 *          IT IS A NEW CONVERSATION, AND NO EXEC CICS                  *
 *          RECEIVE MAP IS ISSUED.                                      *
 *                                                                      *
 *      CP0CP1:                                                         *
 *      EXEC CICS LINK PROGRAM('DSN8CP1') COMMAREA(COMMAREA).         *
 *      UPON RETURN FROM DSN8CP1, EXEC CICS SEND MAP ACCORDING TO     *
 *          THE TYPE SPECIFIED IN PCONVSTA.LASTSCR.                     *
 *      EXEC CICS RETURN TRANSID(D8PS).                                 *
 *                                                                      *
 *  END.                                                                *
 *                                                                      *
 *    *    I.E. LAST CONVERSATION EXISTS, BUT OPERATOR HAD ENTERED    *
 *              DATA FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON  *
 *              SCREEN AND PRESSED ENTER.                               *
 *                                                                      *
 *    **    COMPARM.PFKIN = PF KEY ACTUALLY USED I.E. '01' FOR        *
 *             PF1...                                                   *
 *-------------------------------------------------------------------*/
                                                             %PAGE;
 /*-------------------------------------------------------------------*/
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*                SQL0 CICS  (DSN8CP0)                               */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*                                                                   */
 /*-------------------------------------------------------------------*/
 EXEC SQL INCLUDE DSN8MPCA;                          /*  COMMAREA  */
 EXEC SQL INCLUDE DSN8MPMG;     /*  1ST MAP, BUILT FROM DSN8CPG   */
 EXEC SQL INCLUDE DSN8MPMD;     /*  2ND MAP, BUILT FROM DSN8CPD   */
 EXEC SQL INCLUDE SQLCA;                     /*COMMUNICATION AREA */
 EXEC SQL INCLUDE DSN8MPCS;                          /*  PCONA     */

     /*************************************************************/
     /*       ** DCLGENS AND INITIALIZATIONS                      */
     /*************************************************************/

 DCL STRING BUILTIN;
 DCL J FIXED BIN;
 DCL SAVE_CONVID CHAR(16);
                                        /* DECLARE CONTROL FLAGS */
 DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

     /********************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
     /********************************************************/

 DCL  MODULE            CHAR (07);
 DCL  OUTMSG            CHAR (69);

 DCL DSN8MPG EXTERNAL ENTRY;
0/*******************************************************************/
 /*   SUBMAP REDEFINES THE PL/I STRUCTURE ASSOCIATED WITH THE        */
 /*   CICS MAP DSN8CPD.                                               */
 /*******************************************************************/
```

```
0DCL MAP1PTR PTR,
     MAP2PTR PTR;
 DCL IOAREA AREA(2048);
0DCL 1 SUBMAP(15) BASED (ADDR(DSN8CPDI.LINE1F1L)) UNALIGNED,
        2   COL1LEN      FIXED BIN (15,0) ,
        2   COL1ATTR     CHAR (1) ,
        2   COL1DATA     CHAR (37) ,
        2   COL2LEN      FIXED BIN (15,0) ,
        2   COL2ATTR     CHAR (1) ,
        2   COL2DATA     CHAR (40) ;
0/*******************************************************************/
 /*   PFSTRG IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT    */
 /*   PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID.                  */
 /*******************************************************************/
0DCL CONVID CHAR(16) ;
 DCL PFSTRG CHAR(24) INIT ('123456789:#@ABCDEFGHI>.<') ,
0/*******************************************************************/
 /*   PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS    */
 /*   ALLOWED AS INPUT TO DSN8CP1 AND DSN8CP2 ETC.                    */
 /*******************************************************************/
     PFK(12) CHAR(2) INIT ('01','02','03','04','05','06',
                       '07','08','09','10','11','12' ),
     N FIXED BIN;

0/*******************************************************************/
 /*   SQL RETURN CODE HANDLING                                        */
 /*******************************************************************/
 EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
 EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

0/*******************************************************************/
 /*   ALLOCATE PL/I  WORK AREA / INITIALIZE VARIABLES                 */
 /*******************************************************************/

0ALLOCATE COMMAREA SET(COMMPTR);           /*ALLOCATE COMMON AREA */
 MAP1PTR = ADDR(IOAREA);   /* SET THE POINTER FOR THE GENERAL MAP */
 MAP2PTR = ADDR(IOAREA);   /* SET THE POINTER FOR THE DETAIL  MAP */
 COMMAREA = '' ;                              /*CLEAR COMMON AREA */
 DSN8_MODULE_NAME.MAJOR = 'DSN8CP0 ';         /*GET MODULE NAME   */

                                /*CONSTRUCT CICS CONVERSATION ID */
                                      /*A 4 CHAR TERMINAL ID CON-*/
                                       /*CATENATED WITH 12 BLANKS*/
 CONVID,PCONVSTA.CONVID = EIBTRMID || '            ' ;
 OUTAREA.MAJSYS = 'O';        /*SET MAJOR SYSTEM TO O-ORGANIZATION*/
 EXITCODE = '0';                                /*CLEAR EXIT CODE */

 EXEC CICS HANDLE CONDITION MAPFAIL(CP0SEND);

0/*******************************************************************/
 /*   TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE       */
 /*   LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL.           */
 /*******************************************************************/

0     EXEC SQL SELECT LASTSCR
               INTO   :PCONA.LASTSCR
               FROM   VCONA
               WHERE  CONVID = :CONVID ;

0/*******************************************************************/
 /*   IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO     */
 /*   RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES            */
 /*   TO GET TITLE ETC. FOR OUTPUT MAP.                               */
 /*******************************************************************/

0     IF SQLCODE = +100 THEN GO TO CP0SEND;

0/*******************************************************************/
 /*   IF DATA IS RECEIVED FOR A FIELD, THEN .........MOVE THE DATA    */
 /*   INTO THE CORRESPONDING FIELDS IN INAREA, OTHERWISE MOVE BLANKS. */
 /*                                                                   */
 /*   IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA      */
 /*   FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED     */
 /*   SCREEN AND PRESSED ENTER THEN .........                         */
 /*   MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO         */
 /*   VALIDATION MODULES.                                             */
 /*******************************************************************/

      IF PCONA.LASTSCR = 'DSN8001 ' THEN
        DO;
                                          /*USING LAST SCREEN */
```

```
                                              /*SPECIFIED TO RECEIVE*/
                                               /*INPUT FROM TERMINAL*/
           EXEC CICS RECEIVE MAP ('DSN8CPG') MAPSET ('DSN8CPG') ;

           IF AMAJSYSL ^= 0 THEN    COMPARM.MAJSYS = AMAJSYSI;
                            ELSE    COMPARM.MAJSYS = 'O';
           IF AACTIONL ^= 0 THEN    COMPARM.ACTION = AACTIONI;
                            ELSE    COMPARM.ACTION = ' ';
           IF AOBJECTL ^= 0 THEN    COMPARM.OBJFLD = AOBJECTI;
                            ELSE    COMPARM.OBJFLD = '  ';
           IF ASEARCHL ^= 0 THEN    COMPARM.SEARCH = ASEARCHI;
                            ELSE    COMPARM.SEARCH = '  ';
           IF ADATAL   ^= 0 THEN    COMPARM.DATA   = ADATAI  ;
                            ELSE    COMPARM.DATA = ' ';
        END;

0     ELSE IF PCONA.LASTSCR = 'DSN8002 ' THEN
        DO;
                                           /*MOVE DATA INTO */
                                             /*INPUT FIELDS */
           EXEC CICS RECEIVE MAP ('DSN8CPD') MAPSET('DSN8CPD') ;

           IF BMAJSYSL ^= 0 THEN    COMPARM.MAJSYS = BMAJSYSI;
                            ELSE    COMPARM.MAJSYS = 'O';
           IF BACTIONL ^= 0 THEN    COMPARM.ACTION = BACTIONI;
                            ELSE    COMPARM.ACTION = ' ';
           IF BOBJECTL ^= 0 THEN    COMPARM.OBJFLD = BOBJECTI;
                            ELSE    COMPARM.OBJFLD = '  ';
           IF BSEARCHL ^= 0 THEN    COMPARM.SEARCH = BSEARCHI;
                            ELSE    COMPARM.SEARCH = '  ';
           IF BDATAL   ^= 0 THEN    COMPARM.DATA   = BDATAI  ;
                            ELSE    COMPARM.DATA   = ' ';

           DO I = 1 TO 15;
                IF SUBMAP.COL2LEN(I) ^= 0 THEN
                   COMPARM.TRANDATA(I) = SUBMAP.COL2DATA(I) ;
                ELSE COMPARM.TRANDATA(I) = ' ';
           END;
        END;

0     ELSE                               /* WRONG LASTSCREEN NAME*/
        DO;
           EXEC CICS ABEND ABCODE ('MAPI');
        END;

0/*********************************************************************/
 /*   CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED           */
 /*   BY DSN8CP1 AND DSN8CP2 ETC. EG. PF1 = '01' AND PF13 = '01'.     */
 /*********************************************************************/

0     N = INDEX ( PFSTRG , EIBAID ) ;

      IF N ^= 0 THEN                            /* IF PF KEY USED    */
        DO;
            IF N > 12 THEN N = N - 12 ;         /* PF13 = PF1 ETC.   */
            COMPARM.PFKIN = PFK(N) ;
        END;

      ELSE COMPARM.PFKIN = '  ';                /* IF ENTER | PAKEYS */
      GO TO CP0CP1;

 /*********************************************************************/
 /*                                                                   */
 /*    GO TO DSN8CP1, GET DCLGEN STRUCTURES AND TABLE DCL             */
 /*                                                                   */
 /*********************************************************************/
 CP0SEND :
0          INAREA = ' ' ;                       /*BLANK OUT INAREA */
           COMPARM.PFKIN = '00' ;               /*PUT '00' INTO PFKIN*/

 CP0CP1 :
      INAREA.MAJSYS = 'O';      /*SET MAJOR SYSTEM TO O-ORGANIZATION */

                                                /*GO TO DSN8CP1      */
      EXEC CICS LINK PROGRAM ('DSN8CP1') COMMAREA(COMMAREA)
                                         LENGTH(3000);

0     EXEC SQL INCLUDE DSN8MPXX;              /*GET DCLGEN STRUCTURES*/

 /*********************************************************************/
 /*                                                                   */
 /*    AFTER RETURN FROM DSN8CP1 (SQL1), THE PROGRAM EXAMINES DATA    */
```

```
 /*    PASSED BACK IN PCONVSTA TO SEE WHAT KIND OF SCREEN SHOULD BE   */
 /*    SENT.  PUT THAT DATA INTO THE OUTPUT MAP AND SEND OUTPUT.      */
 /*    IF A SQL ERROR OR WARNING HAD OCCURRED PREVIOUSLY, THE ERROR   */
 /*    MESSAGES ARE EXPECTED TO HAVE BEEN PUT INTO PCONVSTA.          */
 /*                                                                   */
 /*******************************************************************/

      IF PCONVSTA.LASTSCR = 'DSN8001 ' THEN        /*MOVE DATA INTO */
        DO;                                         /*OUTPUT FIELDS */
          ATITLEO = PCONVSTA.TITLE ;
          AMAJSYSO= PCONVSTA.MAJSYS;
          AACTIONO= PCONVSTA.ACTION;
          AOBJECTO= PCONVSTA.OBJFLD;
          ASEARCHO= PCONVSTA.SEARCH;
          ADATAO  = PCONVSTA.DATA  ;
          AMSGO   = PCONVSTA.MSG ;
          ADESCL2O= PCONVSTA.DESC2 ;
          ADESCL3O= PCONVSTA.DESC3 ;
          ADESCL4O= PCONVSTA.DESC4 ;
          APFKEYO = PCONVSTA.PFKTEXT;

          DO I = 1 TO 15;                     /*SEND MAP ACCORDING TO */
               ALINEO(I) = PCONVSTA.OUTPUT.LINE(I); /*PREVIOUS SCREEN*/
          END;

0/*******************************************************************/
 /*   CREATES A DYNAMIC CURSOR                                        */
 /*******************************************************************/
                                        /*SET CURSOR POSITION */
          CURSOR_VALUE = 0;             /*CLEAR CURSOR*/
          IF AACTIONO = ' ' THEN        /*CURSOR SET TO*/
            CURSOR_VALUE = 179;         /*ACTION POSITION*/
          ELSE
            IF AOBJECTO = '  ' THEN     /*CURSOR SET TO*/
              CURSOR_VALUE = 259;       /*OBJFLD POSITION*/
            ELSE
              IF ASEARCHO = '  ' THEN   /*CURSOR SET TO*/
                CURSOR_VALUE = 339;     /*SEARCH CRITERIA POSITION*/
              ELSE
                IF ADATAO = '  ' |
                 ( AACTIONO = 'D' |
                   AACTIONO = 'E' ) THEN    /*CURSOR SET TO */
                       CURSOR_VALUE = 419;   /*DATA POSITION*/

          IF CURSOR_VALUE = 0 THEN           /*SEND OUTPUT MAP */
            EXEC CICS SEND MAP('DSN8CPG') MAPSET('DSN8CPG') ERASE;
          ELSE
            EXEC CICS SEND MAP('DSN8CPG') MAPSET('DSN8CPG') ERASE
                        CURSOR(CURSOR_VALUE);

          IF EXITCODE = '1' THEN             /*  FINISHED ?  */
            EXEC CICS RETURN ; /* EXIT, DON'T REINVOKE TRANSACTION */
          ELSE
            EXEC CICS RETURN TRANSID('D8PS'); /* STANDARD EXIT     */
       END;

0/*******************************************************************/
 /*   MOVES DATA FROM OUTPUT MAP AREA TO                              */
 /*   RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST    */
 /*******************************************************************/
0    ELSE IF PCONVSTA.LASTSCR = 'DSN8002 ' THEN
       DO;
                                              /*MOVE DATA*/
                                              /*FROM OUTPUT FIELDS*/
          BTITLEO = PCONVSTA.TITLE ;
          BMAJSYSO= PCONVSTA.MAJSYS;
          BACTIONO= PCONVSTA.ACTION;
          BOBJECTO= PCONVSTA.OBJFLD;
          BSEARCHO= PCONVSTA.SEARCH;
          BDATAO  = PCONVSTA.DATA  ;
          BMSGO   = PCONVSTA.MSG ;
          BDESCL2O= PCONVSTA.DESC2 ;
          BDESCL3O= PCONVSTA.DESC3 ;
          BDESCL4O= PCONVSTA.DESC4 ;
          BPFKEYO = PCONVSTA.PFKTEXT;

          DO I = 1 TO 15 ;                     /*RECEIVE MAP ACCORDING*/
            SUBMAP.COL1DATA(I) = REOUT.FIELD1(I); /*TO PREVIOUS SCREEN*/
0 /*---------------------------------------------------------------*/
  /*                                                               */
  /*        MODULES DSN8MPE, DSN8MPF ETC. IN SQL2 HAVE PUT THE     */
  /*        ATTRIBUTE BYTE AND CURSOR CONTROL INFO IN IMS MFS      */
```

```
  /*        FORM - HEX'C0' FOR DYNAMIC CURSOR WITH 2 BYTES OF        */
  /*        ATTRIBUTE INFORMATION TO FOLLOW.  THIS PROGRAM CHECKS    */
  /*        FOR THE HEX'C0' AND INSERTS  -1 INTO                     */
  /*        THE LENGTH FIELD ASSOCIATED WITH THE DATA TO CONFORM     */
  /*        WITH THE STANDARD WAY OF HANDLING DYNAMIC CURSORS IN     */
  /*        CICS.  SIMILARLY, ONLY THE SECOND OF THE TWO ATTRIBUTE   */
  /*        BYTES IS MOVED INTO THE CICS ATTRIBUTE BYTE.  THE        */
  /*        FIRST TWO BITS OF THE ATTRIBUTE BYTE IS DIFFERENT        */
  /*        BETWEEN IMS AND CICS STANDARD REPRESENTATIONS, HOWEVER   */
  /*        3270 MANUALS INDICATE THAT ON OUTPUT, THE FIRST          */
  /*        TWO BITS ARE IGNORED.  THUS THE SAME ATTRIBUTE BYTE      */
  /*        IS USED BETWEEN IMS AND CICS MODULES.                    */
  /*                                                               */
  /*--------------------------------------------------------------*/
0           IF UNSPEC(REOUT.ATTR1(I)) = '11000000'B   /*  X'C0' ATTR */
               THEN SUBMAP.COL2LEN(I) = -1;
            SUBMAP.COL2ATTR(I) = REOUT.ATTR2(I);
            SUBMAP.COL2DATA(I) = REOUT.FIELD2(I);
          END;
0/*******************************************************************/
 /*   CREATES A DYNAMIC CURSOR                                      */
 /*******************************************************************/
                                              /*SET CURSOR POSITION */
          CURSOR_VALUE = 0;                   /*CLEAR CURSOR */
          IF BACTIONO = ' ' THEN              /*CURSOR SET TO*/
               CURSOR_VALUE = 179;            /*ACTION POSITION*/
          ELSE
               IF BOBJECTO = '  ' THEN        /*CURSOR SET TO*/
                   CURSOR_VALUE = 259;        /*OBJFLD POSITION*/
               ELSE
                   IF BSEARCHO = '  ' THEN    /*CURSOR SET TO*/
                        CURSOR_VALUE = 339;   /*SEARCH CRITERIA*/
                   ELSE
                       IF BDATAO = '  ' |
                         (BACTIONO = 'D' |
                          BACTIONO = 'E' ) THEN   /*CURSOR SET TO */
                             CURSOR_VALUE = 419;   /*DATA POSITION*/

          IF CURSOR_VALUE = 0 THEN                /*SEND INPUT MAP */
            EXEC CICS SEND MAP('DSN8CPD') MAPSET('DSN8CPD') ERASE;
          ELSE
            EXEC CICS SEND MAP('DSN8CPD') MAPSET('DSN8CPD') ERASE
                        CURSOR(CURSOR_VALUE);

          IF EXITCODE = '1' THEN                     /*FINISHED ? */
            EXEC CICS RETURN ; /* EXIT, DON'T REINVOKE TRANSACTION */
          ELSE
            EXEC CICS RETURN TRANSID('D8PS'); /* STANDARD EXIT     */
       END;
0                 /* SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME */
     ELSE EXEC CICS ABEND ABCODE ('MAPO');
  END;
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CP1

THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE SQL TABLE DCLS AND DCLGEN STRUCURES AS WELL AS PARAMETER AREA.

```
 DSN8CP1:PROC (COMMPTR) OPTIONS(MAIN);
 /*******************************************************************
 *                                                                  *
 *   MODULE NAME = DSN8CP1                                          *
 *                                                                  *
 *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                     *
 *                       SQL 1 MAINLINE                             *
 *                       CICS                                       *
 *                       PL/I                                       *
 *                       ORGANIZATION APPLICATION                   *
 *                                                                  *
 *    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985        *
 *    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083         *
 *                                                                  *
 *    STATUS = RELEASE 2, LEVEL 0                                   *
 *                                                                  *
```

```
*   FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE     *
*              SQL TABLE DCLS AND DCLGEN STRUCURES AS WELL AS        *
*              PARAMETER AREA.                                       *
*              INCLUDE DSN8MP1.                                      *
*              CALL DSN8CP2.                                         *
*              RETURN TO DSN8CP0.                                    *
*                                                                    *
*   NOTES =                                                          *
*      DEPENDENCIES = CALLED BY DSN8CP0, CALLS DSN8CP2 (CICS LINKS). *
*      RESTRICTIONS = NONE                                           *
*                                                                    *
*   MODULE TYPE = PL/I PROC(COMMPTR) OPTIONS.                        *
*      PROCESSOR   = DB2  PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER
*      MODULE SIZE = SEE LINK-EDIT                                   *
*      ATTRIBUTES  = REUSABLE                                        *
*                                                                    *
*   ENTRY POINT = DSN8CP1                                            *
*      PURPOSE = SEE FUNCTION                                        *
*      LINKAGE = NONE                                                *
*                                                                    *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:        *
*                                                                    *
*              SYMBOLIC LABEL/NAME = COMMPTR (POINTER TO COMMAREA)   *
*              DESCRIPTION = NONE                                    *
*                                                                    *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                      *
*                                                                    *
*               SYMBOLIC LABEL/NAME = NONE                           *
*               DESCRIPTION = NONE                                   *
*                                                                    *
*   EXIT-NORMAL = DSN8CP0                                            *
*                                                                    *
*   EXIT-ERROR = DSN8CP0                                             *
*                                                                    *
*      RETURN CODE = NONE                                            *
*                                                                    *
*      ABEND CODES = NONE                                            *
*                                                                    *
*      ERROR-MESSAGES = NONE                                         *
*                                                                    *
*   EXTERNAL REFERENCES =                                            *
*      ROUTINES/SERVICES =   DSN8CP2                                 *
*                                                                    *
*      DATA-AREAS =                                                  *
*         DSN8MPCA            - PLI STRUCTURE FOR COMMAREA           *
*         DSN8MPCS            - DECLARE CONVERSATION STATUS          *
*         DSN8MPOV            - DECLARE OPTION VALIDATION            *
*         DSN8MPVO            - FIND VALID OPTIONS FOR ACTION,       *
*                               OBJECT, SEARCH CRITERIA              *
*         DSN8MP1             - RETRIEVE LAST CONVERSATION,          *
*                               VALIDATE, CALL SQL2                  *
*         DSN8MP3 -- DSN8MP5  - VALIDATION MODULES CALLED BY DSN8MP1 *
*         DSN8MPXX            - SQL ERROR HANDLER                    *
*                                                                    *
*      CONTROL-BLOCKS =                                              *
*         SQLCA               - SQL COMMUNICATION AREA               *
*                                                                    *
*   TABLES =  NONE                                                   *
*                                                                    *
*   CHANGE-ACTIVITY =  NONE                                          *
*                                                                    *
*                                                                    *
*  *PSEUDOCODE*                                                      *
*                                                                    *
*     PROCEDURE                                                      *
*      INCLUDE DECLARATIONS.                                         *
*      INCLUDE DSN8MP1.                                              *
*      INCLUDE ERROR HANDLER.                                        *
*                                                                    *
*      CP1EXIT:  ( REFERENCED BY DSN8MP1 )                           *
*          EXEC CICS RETURN.                                         *
*                                                                    *
*      CP1CALL:  ( REFERENCED BY DSN8MP1 )                           *
*          EXEC CICS LINK PROGRAM('DSN8CP2') COMMAREA(COMMAREA)      *
*                                            LENGTH(3000).           *
*          GO TO MP1SAVE.  (LABEL IN DSN8MP1)                        *
*                                                                    *
*      INCLUDE VALIDATION MODULES.                                   *
*                                                                    *
*   END.                                                             *
*-------------------------------------------------------------------*/
/*------------------------------------------------------------------*/
```

```
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                         SQL1   MAINLINE                           */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*-------------------------------------------------------------------*/
                                        /* SQL RETURN CODE HANDLING */
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

    /****************************************************************/
    /*       ** DCLGENS AND INITIALIZATIONS                         */
    /****************************************************************/

DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
                                        /* DECLARE CONTROL FLAGS */
DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

    /****************************************************/
    /*          **  FIELDS SENT TO MESSAGE ROUTINE         */
    /****************************************************/

DCL  MODULE            CHAR (07);
DCL  OUTMSG            CHAR (69);

DCL DSN8MPG EXTERNAL ENTRY;
EXEC SQL INCLUDE DSN8MPCA;                 /* INCLUDE COMMAREA */
DSN8_MODULE_NAME.MAJOR = 'DSN8CP1 ';       /* INITIALIZE MODULE NAME*/
EXEC SQL INCLUDE DSN8MPCS;                 /* INCLUDE PCONA    */
EXEC SQL INCLUDE DSN8MPOV;                 /* INCLUDE POPTVAL  */
EXEC SQL INCLUDE DSN8MPVO;                 /* INCLUDE CURSOR      */
EXEC SQL INCLUDE SQLCA;                    /* INCLUDE SQL COMMAREA*/
EXEC SQL INCLUDE DSN8MP1;                  /* INCLUDE SQL1 MAIN*/
EXEC SQL INCLUDE DSN8MPXX;                 /* INCLUDE ERRORHANDLER */

CP1EXIT :
     EXEC CICS RETURN;                     /*  STANDARD EXIT   */

CP1CALL :
                                       /* GO TO DSN8CP2 (SQL2)     */
   EXEC CICS LINK PROGRAM('DSN8CP2') COMMAREA(COMMAREA) LENGTH(3000);
   GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3;              /* INCLUDE ACTION VALIDATION*/
EXEC SQL INCLUDE DSN8MP4;              /* INCLUDE OBJECT VALIDATION*/
EXEC SQL INCLUDE DSN8MP5;              /* INCLUDE SEARCH CRITERIA*/
END;                                   /* VALIDATION             */
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CP2

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING CALLS SECONDARY SELECTION MODULES DSN8MPA DSN8MPM CALLS DETAIL MODULES DSN8MPD DSN8MPE DSN8MPF DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ CALLED BY DSN8MP1 (SQL1) .

```
   DSN8CP2: PROC(COMMPTR) OPTIONS(MAIN); /* SQL 2 FOR CICS AND PLI */  00010000
                                                              %PAGE;   00020000
/******************************************************************** 00030000
*                                                                   * 00040000
*   MODULE NAME = DSN8CP2                                           * 00050000
*                                                                   * 00060000
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                      * 00070000
*                      SQL 2 COMMON MODULE                          * 00080000
*                      CICS                                         * 00090000
*                      PL/I                                         * 00100000
*                      ORGANIZATION APPLICATION                     * 00110000
*                                                                   * 00120000
```

```
 *     LICENSED MATERIALS - PROPERTY OF IBM                           * 00130000
 *     5695-DB2                                                       * 00136000
 *     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.       * 00143000
 *                                                                    * 00150000
 *     STATUS = VERSION 4                                             * 00160000
 *                                                                    * 00170000
 *   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING 00180000
 *              CALLS SECONDARY SELECTION MODULES                     * 00190000
 *                    DSN8MPA DSN8MPM                                 * 00200000
 *              CALLS DETAIL MODULES                                  * 00210000
 *                    DSN8MPD DSN8MPE DSN8MPF                         * 00220000
 *                    DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ         * 00230000
 *              CALLED BY DSN8MP1 (SQL1)                              * 00240000
 *                                                                    * 00250000
 *   NOTES =    NONE                                                  * 00260000
 *                                                                    * 00270000
 *   MODULE TYPE = BLOCK OF PL/I CODE                                 * 00280000
 *      PROCESSOR   = DB2  PRECOMPILER, PL/I OPTIMIZER                * 00290000
 *      MODULE SIZE = SEE LINKEDIT                                    * 00300000
 *      ATTRIBUTES  = REUSABLE                                        * 00310000
 *                                                                    * 00320000
 *   ENTRY POINT = DSN8CP2                                            * 00330000
 *      PURPOSE = SEE FUNCTION                                        * 00340000
 *      LINKAGE = NONE                                                * 00350000
 *      INPUT   =                                                     * 00360000
 *                                                                    * 00370000
 *                SYMBOLIC LABEL/NAME = COMMPTR                       * 00380000
 *                DESCRIPTION         = POINTER TO COMMAREA           * 00390000
 *                                      (COMMUNICATION AREA)          * 00400000
 *                                                                    * 00410000
 *      OUTPUT  =                                                     * 00420000
 *                                                                    * 00430000
 *                SYMBOLIC LABEL/NAME = COMMPTR                       * 00440000
 *                DESCRIPTION         = POINTER TO COMMAREA           * 00450000
 *                                      (COMMUNICATION AREA)          * 00460000
 *                                                                    * 00470000
 *   EXIT-NORMAL =                                                    * 00480000
 *                                                                    * 00490000
 *   EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION   * 00500000
 *                SPECIFIED IN DSN8CP2 WILL BE RAISED AND PROGRAM     * 00510000
 *                WILL GO TO THE LABEL DB_ERROR.                      * 00520000
 *                                                                    * 00530000
 *                                                                    * 00540000
 *      RETURN CODE = NONE                                            * 00550000
 *                                                                    * 00560000
 *      ABEND CODES = NONE                                            * 00570000
 *                                                                    * 00580000
 *      ERROR-MESSAGES =                                              * 00590000
 *              DSN8062E-AN OBJECT WAS NOT SELECTED                   * 00600000
 *              DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR               * 00620000
 *              DSN8072E-INVALID SELECTION ON SECONDARY SCREEN        * 00630000
 *                                                                    * 00640000
 *   EXTERNAL REFERENCES = NONE                                       * 00650000
 *      ROUTINES/SERVICES = 10 MODULES LISTED ABOVE                   * 00660000
 *         DSN8MPG             - ERROR MESSAGE ROUTINE                * 00670000
 *                                                                    * 00680000
 *      DATA-AREAS =                                                  * 00690000
 *         DSN8MPA             - SECONDARY SELECTION FOR ORGANIZATION * 00700000
 *         DSN8MPAD            - DECLARE ADMINISTRATIVE DETAIL        * 00710000
 *         DSN8MPAE            - CURSOR EMPLOYEE LIST                 * 00720000
 *         DSN8MPAL            - CURSOR ADMINISTRATION LIST           * 00730000
 *         DSN8MPA2            - DECLARE ADMINISTRATIVE DETAIL        * 00740000
 *         DSN8MPCA            - DECLARE SQL COMMON AREA              * 00750000
 *         DSN8MPD             - DEPARTMENT STRUCTURE DETAIL          * 00760000
 *         DSN8MPDA            - CURSOR ADMINISTRATION LIST           * 00770000
 *         DSN8MPDH            - CURSOR FOR DISPLAY TEXT FROM         * 00780000
 *                               TDSPTXT TABLE                        * 00790000
 *         DSN8MPDM            - DECLARE DEPARTMENT MANAGER           * 00800000
 *         DSN8MPDP            - DELCLARE DEPARTMENT                  * 00810000
 *         DSN8MPDT            - DECLARE DISPLAY TEXT                 * 00820000
 *         DSN8MPE             - DEPARTMENT DETAIL                    * 00830000
 *         DSN8MPEM            - DECLARE EMPLOYEE                     * 00840000
 *         DSN8MPED            - DECLARE EMPLOYEE-DEPARTMENT          * 00845000
 *         DSN8MPF             - EMPLOYEE DETAIL                      * 00850000
 *         DSN8MPOV            - DECLARE OPTION VALIDATION            * 00860000
 *         DSN8MPXX            - ERROR HANDLER                        * 00870000
 *                                                                    * 00880000
 *      CONTROL-BLOCKS =                                              * 00890000
 *         SQLCA               - SQL COMMUNICATION AREA               * 00900000
 *                                                                    * 00910000
 *   TABLES = NONE                                                    * 00920000
 *                                                                    * 00930000
```

```
*   CHANGE-ACTIVITY = NONE                                          *
*                                                                   *
*                                                                   *
*  *PSEUDOCODE*                                                     *
*                                                                   *
*      THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR      *
*      DETAIL MODULE(S) ARE TO BE CALLED IN THE CICS/PL/I           *
*      ENVIRONMENT.                                                 *
*                                                                   *
*      WHAT HAS HAPPENED SO FAR?.............THE SUBSYSTEM          *
*      DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS               *
*      READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL *
*      TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT  *
*      FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF   *
*      ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS     *
*      MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH    *
*      POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE   *
*      BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION     *
*      AND DETAIL MODULES.                                          *
*                                                                   *
*     WHAT IS INCLUDED IN THIS MODULE?.............                 *
*      ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.   *
*      ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE       *
*      SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND               *
*      SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE    *
*      RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE*
*      THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED *
*      IN THIS PROCEDURE.                                           *
*                                                                   *
*  PROCEDURE                                                        *
*         IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR             *
*         IS NOT WILLING TO ACCEPT AN ANSWER THEN                   *
*                         NEW REQUEST*                              *
*                                                                   *
*         ELSE                                                      *
*                IF ANSWER TO A SECONDARY SELECTION THEN            *
*                         DETERMINE IF NEW REQUEST.                 *
*                                                                   *
*                                                                   *
*         CASE (NEW REQUEST)                                        *
*                                                                   *
*                SUBCASE ('ADD')                                    *
*                        DETAIL PROCESSOR                           *
*                        RETURN TO SQL 1                            *
*                ENDSUB                                             *
*                                                                   *
*                SUBCASE ('ERASE','DISPLAY','UPDATE')               *
*                        CALL SECONDARY SELECTION                   *
*                        IF # OF POSSIBLE CHOICES IS ^= 1 THEN      *
*                               RETURN TO SQL 1                     *
*                        ELSE                                       *
*                               CALL THE DETAIL PROCESSOR           *
*                               RETURN TO SQL 1                     *
*                 ENDSUB                                            *
*                                                                   *
*          ENDCASE                                                  *
*                                                                   *
*         IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS      *
*         ACTUALLY BEEN MADE THEN                                   *
*                     VALID SELECTION #?                            *
*                     IF IT IS VALID THEN                           *
*                           CALL DETAIL PROCESSOR                   *
*                           RETURN TO SQL 1                         *
*                     ELSE                                          *
*                           PRINT ERROR MSG                         *
*                           RETURN TO SQL 1.                        *
*                                                                   *
*         IF ANSWER TO SECONDARY SELECTION THEN                     *
*                  CALL SECONDARY SELECTION                         *
*                  RETURN TO SQL 1.                                 *
*                                                                   *
*         IF ANSWER TO DETAIL THEN                                  *
*                  CALL DETAIL PROCESSOR                            *
*                  RETURN TO SQL 1.                                 *
*                                                                   *
*   END.                                                            *
*                                                                   *
*        *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES*
*         THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.       *
*------------------------------------------------------------------*/

  /* INCLUDE DECLARES */
```

```
   EXEC SQL INCLUDE DSN8MPCA;   /*COMMUNICATION AREA BETWEEN MODULES */
   EXEC SQL INCLUDE SQLCA;      /*SQL COMMUNICATION AREA            */
                               /* ORGANIZATION                      */
   EXEC SQL INCLUDE DSN8MPDP;   /* DCLGEN FOR DEPARTMENT            */
   EXEC SQL INCLUDE DSN8MPEM;   /* DCLGEN FOR EMPLOYEE              */
   EXEC SQL INCLUDE DSN8MPED;   /* DCLGEN FOR EMPLOYEE-DEPARTMENT   */
   EXEC SQL INCLUDE DSN8MPDM;   /* DCLGEN FOR DEPARTMENT/MANAGER    */
   EXEC SQL INCLUDE DSN8MPAD;   /* DCLGEN FOR ADMINISTRATION DETAIL */
   EXEC SQL INCLUDE DSN8MPA2;   /* DCLGEN FOR ADMINISTRATION DETAIL */
                               /* PROGRAMMING TABLES                */
   EXEC SQL INCLUDE DSN8MPOV;   /* DCLGEN FOR OPTION VALIDATION     */
   EXEC SQL INCLUDE DSN8MPDT;   /* DCLGEN FOR DISPLAY TEXT TABLE    */

    /* CURSORS */
   EXEC SQL INCLUDE DSN8MPAL;    /* MAJSYS O - SEC SEL FOR DS AND DE */
   EXEC SQL INCLUDE DSN8MPAE;    /* MAJSYS O - SEC SEL FOR EM        */
   EXEC SQL INCLUDE DSN8MPDA;    /* MAJSYS O - DETAIL  FOR DS        */
   EXEC SQL INCLUDE DSN8MPDH;    /* PROG TABLES - DISPLAY HEADINGS   */

   DCL VERIFY BUILTIN;
   DCL UNSPEC BUILTIN;
   DCL DSN8MPG EXTERNAL ENTRY;

     /***********************************************************/
     /*       ** DCLGENS AND INITIALIZATIONS                    */
     /***********************************************************/

 DCL STRING BUILTIN;
 DCL J FIXED BIN;
 DCL SAVE_CONVID CHAR(16);
                                          /* DECLARE CONTROL FLAGS */
 DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

  /*******************************************/
  /* FIELDS SENT TO MESSAGE ROUTINE          */
  /*******************************************/

    DCL MODULE        CHAR (07) INIT('DSN8CP2');
    DCL OUTMSG        CHAR (69);

  /*****************************************************/
  /* SQL RETURN CODE HANDLING                          */
  /*****************************************************/

   EXEC SQL WHENEVER SQLERROR   GO TO DB_ERROR;
   EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

  /*****************************************************/
  /*   INITIALIZATIONS                                 */
  /*****************************************************/

   DSN8_MODULE_NAME.MAJOR='DSN8CP2';
   DSN8_MODULE_NAME.MINOR=' ';

     /*****************************************************/
     /*  DETERMINES WHETHER NEW REQUEST OR NOT            */
     /*****************************************************/

  /* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */
  /* NEW REQUEST.                                                 */

   IF PCONVSTA.PREV = ' ' THEN
     COMPARM.NEWREQ = 'Y';

  /* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A  */
  /* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/
  /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */

   IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' &
      SUBSTR(COMPARM.DATA,3,58) ^= ' ' &
      COMPARM.PFKIN ^= '08'
      THEN COMPARM.NEWREQ = 'Y';

     /*****************************************************/
     /*  IF NEW REQUEST AND ACTION IS 'ADD' THEN          */
     /*       CALL DETAIL PROCESSOR                       */
     /*  ELSE CALL SECONDARY SELECTION                    */
     /*****************************************************/

   IF COMPARM.NEWREQ='Y' THEN
     DO;
```

```
     IF COMPARM.ACTION = 'A' THEN
       DO;
         CALL DETAIL;                    /*CALL DETAIL PROCESSOR */
         GO TO EXIT;                     /* RETURN               */
       END;

     CALL SECSEL;                        /*CALL SECONDARY SELECTION*/

     IF MAXSEL = 1 THEN                  /* IF NO. OF CHOICES = 1  */
       CALL DETAIL;                      /* CALL DETAIL PROCESSOR  */
     GO TO EXIT;                         /* RETURN                 */
   END;

/* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */
/* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS                */
/* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2     */
/* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION.            */

   /*********************************************************/
   /*  DETERMINES IF VALID SELECTION NUMBER                 */
   /*********************************************************/

 IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */

 IF PCONVSTA.MAXSEL < 1  THEN GO TO IP201; /* NO VALID CHOICES */

 IF COMPARM.PFKIN = '08' THEN GO TO IP201; /* SCROLL REQUEST   */

 IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2)
                THEN GO TO IP201;  /* NO CHANGE ON INPUT SCREEN */

 IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN     /* SECOND CHAR BLANK */
    IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN
      DO;
        SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1);
        SUBSTR(COMPARM.DATA,1,1) = '0';
      END;

 IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 &
           SUBSTR(COMPARM.DATA,1,2) > '00' THEN

       IF SUBSTR(COMPARM.DATA,1,2) <= PCONVSTA.MAXSEL THEN
         DO;
           COMPARM.NEWREQ = 'Y';  /*TELL DETAIL PROCESSOR NEW REQ */
           CALL DETAIL;                  /* CALL DETAIL PROCESSOR*/
           GO TO EXIT;                   /* RETURN*/
         END;

                                       /*INVALID SELECTION NO.*/
                                       /*PRINT ERROR MESSAGE */
        CALL DSN8MPG (MODULE, '072E', OUTMSG);
        PCONVSTA.MSG= OUTMSG;


 GO TO EXIT;                           /*    RETURN          */

   /*********************************************************/
   /*  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL      */
   /*********************************************************/

/* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */
IP201:

 IF PCONVSTA.PREV = 'S' THEN
   DO;
     CALL SECSEL;                           /*SECONDARY SELECTION*/
     GO TO EXIT;                            /* RETURN            */
   END;

 IF PCONVSTA.PREV = 'D' THEN
   DO;
     CALL DETAIL;                            /* DETAIL PROCESSOR  */
     GO TO EXIT;                             /* RETURN            */
   END;

                                             /*LOGIC ERROR        */
 CALL DSN8MPG (MODULE, '066E', OUTMSG);
 PCONVSTA.MSG= OUTMSG;                       /*PRINT ERROR MESSAGE*/
 GO TO EXIT;

  EXEC SQL INCLUDE DSN8MPXX;                  /*HANDLES SQL ERRORS*/
  GO TO EXIT;
```

```

    /**********************************************************/
    /*  CALLS SECONDARY SELECTION AND RETURNS TO SQL 1        */
    /* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
    /**********************************************************/

SECSEL: PROC;                    /*CALL APPROPRIATE SECONDARY SEL   */
      PCONVSTA.LASTSCR = 'DSN8001';          /* NOTE GENERAL MAP    */

      IF COMPARM.OBJFLD='DS' THEN            /*ADMINISTRATIVE       */
        DO;                                  /*DEPARTMENT STRUCTURE */
          CALL DSN8MPA;
          RETURN;
        END;

      IF COMPARM.OBJFLD='DE' THEN            /*INDIVIDUAL DEPARTMENT*/
        DO;                                  /*PROCESSING           */
          CALL DSN8MPA;
          RETURN;
        END;

      IF COMPARM.OBJFLD='EM' THEN             /*INDIVIDUAL EMPLOYEE */
        DO;                                   /*PROCESSING          */
          CALL DSN8MPA;
          RETURN;
        END;
                                             /*MISSING SECONDARY SEL*/
                                             /*PRINT ERROR MESSAGE  */
  CALL DSN8MPG (MODULE, '062E', OUTMSG);
  PCONVSTA.MSG= OUTMSG;                        /*PRINT ERROR MESSAGE*/

  GO TO EXIT;                                 /*RETURN */
  END SECSEL;

     /*********************************************************/
     /* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1           */
     /*********************************************************/

DETAIL: PROC;                 /* CALL APPROPRIATE DETAIL MODULE */
        PCONVSTA.LASTSCR = 'DSN8002';    /* NOTE DETAIL  MAP    */

        SELECT (COMPARM.OBJFLD);

          WHEN('DS') CALL DSN8MPD;       /*DEPARTMENT STRUCTURE */

          WHEN('DE') CALL DSN8MPE;       /*DEPARTMENT*/

          WHEN('EM') CALL DSN8MPF;       /*EMPLOYEE*/

                                         /*MISSING DETAIL MODULE*/
          OTHERWISE                     /*PRINT ERROR MESSAGE  */
            DO;
              CALL DSN8MPG (MODULE, '062E', OUTMSG);
              PCONVSTA.MSG= OUTMSG;
            END;
        END;
  END DETAIL;

                                         /*RETURNS TO SQL 1*/
 EXIT: EXEC CICS RETURN;

       EXEC SQL INCLUDE DSN8MPA;       /* SEC SEL - ADMIN STRUCTURE */
       EXEC SQL INCLUDE DSN8MPD;       /* DETAIL  - ADMIN STRUCTURE */
       EXEC SQL INCLUDE DSN8MPE;       /* DETAIL  - DEPARTMENTS */
       EXEC SQL INCLUDE DSN8MPF;       /* DETAIL  - EMPLOYEES   */
  END DSN8CP2;
```

Related reference

"Sample applications in CICS" on page 1378

A set of Db2 sample applications run in the CICS environment.

DSN8CP6

THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE INPUT, CALLS DSN8CP7, AND ISSUES A CICS SEND MAP AFTER RETURNING.

```
DSN8CP6 : PROC OPTIONS (MAIN);
/***********************************************************************
```

```
*                                                                       * 00030000
*   MODULE NAME = DSN8CP6                                               * 00040000
*                                                                       * 00050000
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                          * 00060000
*                      SUBSYSTEM INTERFACE MODULE                       * 00070000
*                      CICS                                             * 00080000
*                      PL/I                                             * 00090000
*                      PROJECT APPLICATION                              * 00100000
*                                                                       * 00110000
*     LICENSED MATERIALS - PROPERTY OF IBM 5605-DB2                     * 00120000
*     (C) COPYRIGHT 1982, 2010 IBM CORP.  ALL RIGHTS RESERVED.          * 00130000
*                                                                       * 00140000
*     STATUS = VERSION 10                                               * 00150000
*                                                                       * 00160000
*   FUNCTION = THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE        * 00170000
*              INPUT, CALLS DSN8CP7, AND ISSUES A CICS SEND             * 00180000
*               MAP AFTER RETURNING.                                    * 00190000
*   NOTES =                                                             * 00200000
*         1.INITIALIZES ITSELF WHEN TERMINAL OPERATOR ENTER INPUT       * 00210000
*           AFTER VIEWING THE SCREEN SENT BY THE PREVIOUS               * 00220000
*           ITERATION OF THE PROGRAM.                                   * 00230000
*                                                                       * 00240000
*      DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED :              * 00250000
*                              DSN8MCME AND DSN8MCMF.                   * 00260000
*                     MODULES DSN8CP7 IS REQUIRED.                      * 00270000
*                     DCLGEN STRUCTURE DSN8MPCS IS REQUIRED.            * 00280000
*                     INCLUDED PLI STRUCTURE DSN8MPCA IS REQUIRED.      * 00290000
*                                                                       * 00300000
*      RESTRICTIONS = NONE                                              * 00310000
*                                                                       * 00320000
*                                                                       * 00330000
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                               * 00340000
*      PROCESSOR   = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER* 00350000
*      MODULE SIZE = SEE LINK-EDIT                                      * 00360000
*      ATTRIBUTES  = REUSABLE                                           * 00370000
*                                                                       * 00380000
*   ENTRY POINT = DSN8CP6                                               * 00390000
*      PURPOSE = SEE FUNCTION                                           * 00400000
*      LINKAGE = NONE                                                   * 00410000
*        INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:         * 00420000
*                                                                       * 00430000
*                SYMBOLIC LABEL/NAME = NONE                             * 00440000
*                DESCRIPTION = NONE                                     * 00450000
*                                                                       * 00460000
*       OUTPUT = PARAMETERS EXPLICITLY RETURNED:                        * 00470000
*                                                                       * 00480000
*                SYMBOLIC LABEL/NAME = NONE                             * 00490000
*                DESCRIPTION = NONE                                     * 00500000
*                                                                       * 00510000
*                                                                       * 00520000
*   EXIT-NORMAL = CICS RETURN TRANSID(D8PP).                            * 00530000
*                                                                       * 00540000
*   EXIT-ERROR = DB_ERROR FOR SQL ERRORS.                               * 00550000
*                CICS ABEND FOR CICS PROBLEMS.                          * 00560000
*                NO PL/I ON CONDITIONS.                                 * 00570000
*                                                                       * 00580000
*      RETURN CODE = NONE                                               * 00590000
*                                                                       * 00600000
*      ABEND CODES =  NONE                                              * 00610000
*                                                                       * 00620000
*      ERROR-MESSAGES = NONE                                            * 00630000
*                                                                       * 00640000
*   EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS                      * 00650000
*      ROUTINES/SERVICES = DSN8CP7                                      * 00660000
*                                                                       * 00670000
*      DATA-AREAS =                                                     * 00680000
*         DSN8MPCA            - PARAMETER TO BE PASSED TO DSN8CP7       * 00690000
*                               COMMON AREA                             * 00700000
*         DSN8MPCS            - DECLARE CONVERSATION STATUS             * 00710000
*         DSN8MPMF            - CICS/OS/VS PL/I MAP, PROJECTS           * 00720000
*         DSN8MPME            - CICS/OS/VS PL/I MAP, PROJECTS           * 00730000
*                                                                       * 00740000
*      CONTROL-BLOCKS =                                                 * 00750000
*         SQLCA               - SQL COMMUNICATION AREA                  * 00760000
*                                                                       * 00770000
*   TABLES =  NONE                                                      * 00780000
*                                                                       * 00790000
*   CHANGE-ACTIVITY =  NONE                                             * 00800000
*                                                                       * 00810000
*                                                                       * 00820000
*  *PSEUDOCODE*                                                         * 00830000
*                                                                       * 00840000
```

```
 *      PROCEDURE                                                      *
 *      DECLARATIONS.                                                  *
 *      ALLOCATE PLI WORK AREA FOR COMMAREA.                           *
 *         PUT MODULE NAME 'DSN8CP6' IN AREA USED BY ERROR-HANDLER.   *
 *         PUT CICS EIBTRMID IN PCONVSTA.CONVID TO BE PASSED TO DSN8CP7
 *         RETRIEVE LASTSCR FROM VCONA USING THE CONVID TO DETERMINE  *
 *         WHICH OF THE TWO BMS MAPS SHOULD BE USED TO MAP IN DATA.   *
 *                                                                     *
 *      IF RETRIEVAL IS SUCCESSFUL, THEN DO.                           *
 *          EXEC CICS RECEIVE MAP ACCORDING TO SPECIFIED LASTSCR      *
 *          IF MAPFAIL CONDITION IS RAISED* THEN DO.                   *
 *                  COMPARM.PFKIN = '00'                               *
 *                  GO TO CP6CP7                                       *
 *                  END                                                *
 *          ELSE                                                       *
 *                  PUT DATA FROM MAP INTO COMPARM **                  *
 *                                                                     *
 *      ELSE                                                           *
 *          IT IS A NEW CONVERSATION,                                  *
 *          AND NO EXEC CICS RECEIVE MAP IS ISSUED.                    *
 *                                                                     *
 *      CP6CP7:                                                        *
 *      EXEC CICS LINK PROGRAM('DSN8CP7') COMMAREA(COMMAREA).         *
 *      UPON RETURN FROM DSN8CP7, EXEC CICS SEND MAP ACCORDING TO     *
 *          THE TYPE SPECIFIED IN PCONVSTA.LASTSCR.                    *
 *      EXEC CICS RETURN TRANSID(D8PP).                                *
 *                                                                     *
 *   END.                                                              *
 *                                                                     *
 *    *    I.E. LAST CONVERSATION EXISTS, BUT OPERATOR HAD ENTERED    *
 *              DATA FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON  *
 *              SCREEN AND PRESSED ENTER.                              *
 *                                                                     *
 *    **    COMPARM.PFKIN = PF KEY ACTUALLY USED I.E. '01' FOR        *
 *             PF1...                                                  *
 /*-----------------------------------------------------------------*/
 /*                                                                 */
 /*                                                                 */
 /*                                                                 */
 /*                                                                 */
 /*                 SQL0 CICS  (DSN8CP6)                            */
 /*                                                                 */
 /*                                                                 */
 /*                                                                 */
 /*                                                                 */
 /*                                                                 */
 /*-----------------------------------------------------------------*/
 EXEC SQL INCLUDE DSN8MPCA;                       /*  COMMAREA  */
 EXEC SQL INCLUDE DSN8MPMF;   /*  1ST MAP, BUILT FROM DSN8CPF     */
 EXEC SQL INCLUDE DSN8MPME;   /*  2ND MAP, BUILT FROM DSN8CPE     */
 EXEC SQL INCLUDE SQLCA;            /*  SQL COMMUNICATION AREA    */
 EXEC SQL INCLUDE DSN8MPCS;                       /*  PCONA     */

0/*******************************************************************/
 /*   SUBMAP REDEFINES THE PL/I STRUCTURE ASSOCIATED WITH THE        */
 /*   CICS MAP DSN8CPE.                                              */
 /*******************************************************************/

0DCL MAP1PTR PTR,
     MAP2PTR PTR;
 DCL IOAREA AREA(2048);
0DCL 1 SUBMAP(15) BASED (ADDR(DSN8CPEI.LINE1F1L)) UNALIGNED,
        2   COL1LEN      FIXED BIN (15,0) ,
        2   COL1ATTR     CHAR (1) ,
        2   COL1DATA     CHAR (37) ,
        2   COL2LEN      FIXED BIN (15,0) ,
        2   COL2ATTR     CHAR (1) ,
        2   COL2DATA     CHAR (40) ;

0/*******************************************************************/
 /*   PFSTRG IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT    */
 /*   PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID.                  */
 /*******************************************************************/

0DCL CONVID CHAR(16) ;
 DCL PFSTRG CHAR(24) INIT ('123456789:#@ABCDEFGHI>.<') ,

0/*******************************************************************/
 /*   PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS    */
 /*   ALLOWED AS INPUT TO DSN8CP7 AND DSN8CP8 ETC.                    */
 /*******************************************************************/
```

```
     PFK(12) CHAR(2) INIT ('01','02','03','04','05','06',
                           '07','08','09','10','11','12' ),
     N FIXED BIN;

     /****************************************************************/
     /*       ** DCLGENS AND INITIALIZATIONS                        */
     /****************************************************************/

 DCL STRING BUILTIN;
 DCL J FIXED BIN;
 DCL SAVE_CONVID CHAR(16);
                                     /* DECLARE CONTROL FLAGS */
 DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

     /********************************************************/
     /*          **  FIELDS SENT TO MESSAGE ROUTINE          */
     /********************************************************/

 DCL  MODULE            CHAR (07);
 DCL  OUTMSG            CHAR (69);

 DCL DSN8MPG EXTERNAL ENTRY;

0/*********************************************************************/
 /*   SQL RETURN CODE HANDLING                                        */
 /*********************************************************************/

 EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
 EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

0/*********************************************************************/
 /*   ALLOCATE PL/I  WORK AREA / INITIALIZE VARIABLES                 */
 /*********************************************************************/

0ALLOCATE COMMAREA SET(COMMPTR);           /*ALLOCATE COMMON AREA */
 MAP1PTR = ADDR(IOAREA);   /* SET THE POINTER FOR THE GENERAL MAP */
 MAP2PTR = ADDR(IOAREA);   /* SET THE POINTER FOR THE DETAIL  MAP */
 COMMAREA = '' ;                              /*CLEAR COMMON AREA */
 DSN8_MODULE_NAME.MAJOR = 'DSN8CP6 ';         /*GET MODULE NAME   */
                                  /*CONSTRUCT CICS CONVERSATION ID */
                                        /*A 4 CHAR TERMINAL ID CON-*/
                                         /*CATENATED WITH 12 BLANKS*/
  CONVID,PCONVSTA.CONVID = EIBTRMID || '            ' ;
  OUTAREA.MAJSYS = 'P';        /*SET MAJOR SYSTEM TO P-PROJECT    */
  EXITCODE = '0';                               /*CLEAR EXIT CODE */


      EXEC CICS HANDLE CONDITION MAPFAIL(CP6SEND);
0/*********************************************************************/
 /*   TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE       */
 /*   LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL.           */
 /*********************************************************************/

0     EXEC SQL SELECT LASTSCR
               INTO   :PCONA.LASTSCR
               FROM   VCONA
               WHERE  CONVID = :CONVID ;

0/*********************************************************************/
 /*   IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO     */
 /*   RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES            */
 /*   TO GET TITLE ETC. FOR OUTPUT MAP.                               */
 /*********************************************************************/

0     IF SQLCODE = +100 THEN GO TO CP6SEND;

0/*********************************************************************/
 /*   IF DATA IS RECEIVED FOR A FIELD, THEN .........MOVE THE DATA    */
 /*   INTO THE CORRESPONDING FIELDS IN INAREA, OTHERWISE MOVE BLANKS. */
 /*                                                                   */
 /*   IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA      */
 /*   FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED     */
 /*   SCREEN AND PRESSED ENTER THEN .........                         */
 /*   MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO         */
 /*   VALIDATION MODULES.                                             */
 /*********************************************************************/

      IF PCONA.LASTSCR = 'DSN8001 ' THEN
        DO;
                                              /*USING LAST SCREEN */
                                              /*SPECIFIED TO RECEIVE*/
                                              /*INPUT FROM TERMINAL*/
```

```
            EXEC CICS RECEIVE MAP ('DSN8CPF') MAPSET ('DSN8CPF') ;
            IF AMAJSYSL ^= 0 THEN    COMPARM.MAJSYS = AMAJSYSI;
                             ELSE    COMPARM.MAJSYS = 'P';
            IF AACTIONL ^= 0 THEN    COMPARM.ACTION = AACTIONI;
                             ELSE    COMPARM.ACTION = ' ';
            IF AOBJECTL ^= 0 THEN    COMPARM.OBJFLD = AOBJECTI;
                             ELSE    COMPARM.OBJFLD = '  ';
            IF ASEARCHL ^= 0 THEN    COMPARM.SEARCH = ASEARCHI;
                             ELSE    COMPARM.SEARCH = '  ';
            IF ADATAL   ^= 0 THEN    COMPARM.DATA   = ADATAI  ;
                             ELSE    COMPARM.DATA = ' ';
        END;

0    ELSE IF PCONA.LASTSCR = 'DSN8002 ' THEN
        DO;
                                            /*MOVE DATA INTO */
                                              /*INPUT FIELDS */
            EXEC CICS RECEIVE MAP ('DSN8CPE') MAPSET('DSN8CPE') ;
            IF BMAJSYSL ^= 0 THEN    COMPARM.MAJSYS = BMAJSYSI;
                             ELSE    COMPARM.MAJSYS = 'P';
            IF BACTIONL ^= 0 THEN    COMPARM.ACTION = BACTIONI;
                             ELSE    COMPARM.ACTION = ' ';
            IF BOBJECTL ^= 0 THEN    COMPARM.OBJFLD = BOBJECTI;
                             ELSE    COMPARM.OBJFLD = '  ';
            IF BSEARCHL ^= 0 THEN    COMPARM.SEARCH = BSEARCHI;
                             ELSE    COMPARM.SEARCH = '  ';
            IF BDATAL   ^= 0 THEN    COMPARM.DATA   = BDATAI  ;
                             ELSE    COMPARM.DATA   = ' ';

            DO I = 1 TO 15;
                 IF SUBMAP.COL2LEN(I) ^= 0 THEN
                    COMPARM.TRANDATA(I) = SUBMAP.COL2DATA(I) ;
                 ELSE COMPARM.TRANDATA(I) = ' ';
            END;
        END;

0     ELSE                                   /* WRONG LASTSCREEN NAME*/
         DO;
            EXEC CICS ABEND ABCODE ('MAPI');
         END;

0/*******************************************************************/
 /*   CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED         */
 /*   BY DSN8CP7 AND DSN8CP8 ETC. EG. PF1 = '01' AND PF13 = '01'.    */
 /*******************************************************************/

0     N = INDEX ( PFSTRG , EIBAID ) ;
      IF N ^= 0 THEN                             /* IF PF KEY USED    */
        DO;
             IF N > 12 THEN N = N - 12 ;         /* PF13 = PF1 ETC.   */
             COMPARM.PFKIN = PFK(N) ;
        END;

      ELSE COMPARM.PFKIN = '  ';                 /* IF ENTER | PAKEYS */
      GO TO CP6CP7;

 /*******************************************************************/
 /*                                                                 */
 /*    GO TO DSN8CP7, GET DCLGEN STRUCTURES AND TABLE DCL           */
 /*                                                                 */
 /*******************************************************************/

  CP6SEND:
      INAREA = ' ' ;                            /*BLANK OUT INAREA */
      COMPARM.PFKIN = '00' ;                  /*PUT '00' INTO PFKIN*/

  CP6CP7 :
      INAREA.MAJSYS = 'P';        /*SET MAJOR SYSTEM TO P-PROJECT */

                                                /* GO TO DSN8CP7   */
      EXEC CICS LINK PROGRAM ('DSN8CP7') COMMAREA(COMMAREA)
                                  LENGTH(3000);

0     EXEC SQL INCLUDE DSN8MPXX;            /*GET DCLGEN STRUCTURES*/

                                                                   %PAGE;
 /*******************************************************************/
 /*                                                                 */
 /*    AFTER RETURN FROM DSN8CP7 (SQL1), THE PROGRAM EXAMINES DATA    */
 /*    PASSED BACK IN PCONVSTA TO SEE WHAT KIND OF SCREEN SHOULD BE   */
 /*    SENT.  PUT THAT DATA INTO THE OUTPUT MAP AND SEND OUTPUT.      */
 /*    IF A SQL ERROR OR WARNING HAD OCCURRED PREVIOUSLY, THE ERROR   */
```

```
 /*    MESSAGES ARE EXPECTED TO HAVE BEEN PUT INTO PCONVSTA.          */
 /*                                                                   */
 /*******************************************************************/

      IF PCONVSTA.LASTSCR = 'DSN8001 ' THEN      /*MOVE DATA INTO */
        DO;                                      /*OUTPUT FIELDS */
          ATITLEO = PCONVSTA.TITLE ;
          AMAJSYSO= PCONVSTA.MAJSYS;
          AACTIONO= PCONVSTA.ACTION;
          AOBJECTO= PCONVSTA.OBJFLD;
          ASEARCHO= PCONVSTA.SEARCH;
          ADATAO  = PCONVSTA.DATA  ;
          AMSGO   = PCONVSTA.MSG ;
          ADESCL2O= PCONVSTA.DESC2 ;
          ADESCL3O= PCONVSTA.DESC3 ;
          ADESCL4O= PCONVSTA.DESC4 ;
          APFKEYO = PCONVSTA.PFKTEXT;

          DO I = 1 TO 15;                    /*SEND MAP ACCORDING TO*/
               ALINEO(I) = PCONVSTA.OUTPUT.LINE(I); /*PREVIOUS SCREEN*/
          END;

0/*******************************************************************/
 /*   CREATES A DYNAMIC CURSOR                                       */
 /*******************************************************************/

                                             /*SET CURSOR POSITION */
          CURSOR_VALUE = 0;                         /*CLEAR CURSOR */
          IF AACTIONO = ' ' THEN                    /*CURSOR SET TO*/
               CURSOR_VALUE = 179;                /*ACTION POSITION*/
          ELSE
               IF AOBJECTO = '  ' THEN               /*CURSOR SET TO*/
                   CURSOR_VALUE = 259;             /*OBJECT POSITION*/
               ELSE
                   IF ASEARCHO = '  ' THEN           /*CURSOR SET TO*/
                        CURSOR_VALUE = 339;        /*SEARCH POSITION*/
                   ELSE
                        IF  ADATAO = '  ' |         /*CURSOR SET TO*/
                           (AACTIONO = 'D' |        /*DATA POSITION*/
                            AACTIONO = 'U' |
                            AACTIONO = 'A' |
                            AACTIONO = 'E' ) THEN
                                 CURSOR_VALUE = 419;


          IF CURSOR_VALUE = 0 THEN                 /*SEND OUTPUT MAP */
            EXEC CICS SEND MAP('DSN8CPF') MAPSET('DSN8CPF');
          ELSE
            EXEC CICS SEND MAP('DSN8CPF') MAPSET('DSN8CPF') ERASE
                  CURSOR(CURSOR_VALUE);

          IF EXITCODE = '1' THEN                        /*FINISHED ? */
            EXEC CICS RETURN;   /* RETURN, DON'T REINVOKE TRANSACTION*/
          ELSE
            EXEC CICS RETURN TRANSID('D8PP');     /* STANDARD RETURN */
      END;

0/*******************************************************************/
 /*   MOVES DATA FROM OUTPUT MAP AREA TO                             */
 /*   RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST   */
 /*******************************************************************/

                                                      /*MOVE DATA*/
                                               /*FROM OUTPUT FIELDS*/
0    ELSE IF PCONVSTA.LASTSCR = 'DSN8002 ' THEN
       DO;
          BTITLEO = PCONVSTA.TITLE ;
          BMAJSYSO= PCONVSTA.MAJSYS;
          BACTIONO= PCONVSTA.ACTION;
          BOBJECTO= PCONVSTA.OBJFLD;
          BSEARCHO= PCONVSTA.SEARCH;
          BDATAO  = PCONVSTA.DATA  ;
          BMSGO   = PCONVSTA.MSG ;
          BDESCL2O= PCONVSTA.DESC2 ;
          BDESCL3O= PCONVSTA.DESC3 ;
          BDESCL4O= PCONVSTA.DESC4 ;
          BPFKEYO = PCONVSTA.PFKTEXT;

          DO I = 1 TO 15 ;              /*RECEIVE MAP ACCORDING TO */
            SUBMAP.COL1DATA(I) = REOUT.FIELD1(I); /*PREVIOUS SCREEN */

0 /*-----------------------------------------------------------------*/
```

```
  /*                                                              */
  /*        MODULES DSN8MPE, DSN8MPF ETC. IN SQL2 HAVE PUT THE      */
  /*        ATTRIBUTE BYTE AND CURSOR CONTROL INFO IN IMS MFS       */
  /*        FORM - HEX'C0' FOR DYNAMIC CURSOR WITH 2 BYTES OF       */
  /*        ATTRIBUTE INFORMATION TO FOLLOW.  THIS PROGRAM CHECKS   */
  /*        FOR THE HEX'C0' AND INSERTS  -1 INTO                    */
  /*        THE LENGTH FIELD ASSOCIATED WITH THE DATA TO CONFORM    */
  /*        WITH THE STANDARD WAY OF HANDLING DYNAMIC CURSORS IN    */
  /*        CICS.  SIMILARLY, ONLY THE SECOND OF THE TWO ATTRIBUTE  */
  /*        BYTES IS MOVED INTO THE CICS ATTRIBUTE BYTE.  THE       */
  /*        FIRST TWO BITS OF THE ATTRIBUTE BYTE IS DIFFERENT       */
  /*        BETWEEN IMS AND CICS STANDARD REPRESENTATIONS, HOWEVER  */
  /*        3270 MANUALS INDICATE THAT ON OUTPUT, THE FIRST         */
  /*        TWO BITS ARE IGNORED.  THUS THE SAME ATTRIBUTE BYTE     */
  /*        IS USED BETWEEN IMS AND CICS MODULES.                   */
  /*                                                              */
  /*--------------------------------------------------------------*/

0           IF UNSPEC(REOUT.ATTR1(I)) = '11000000'B   /*  X'C0' ATTR */
               THEN SUBMAP.COL2LEN(I) = -1;
            SUBMAP.COL2ATTR(I) = REOUT.ATTR2(I);
            SUBMAP.COL2DATA(I) = REOUT.FIELD2(I);
          END;

0/*****************************************************************/
 /*   CREATES A DYNAMIC CURSOR                                    */
 /*****************************************************************/

                                                /*SET CURSOR POSITION */
          CURSOR_VALUE = 0;                          /*CLEAR CURSOR */
          IF BACTIONO = ' ' THEN                     /*CURSOR SET TO*/
               CURSOR_VALUE = 179;                 /*ACTION POSITION*/
          ELSE
               IF BOBJECTO = '  ' THEN               /*CURSOR SET TO*/
                    CURSOR_VALUE = 259;            /*OBJECT POSITION*/
               ELSE
                    IF BSEARCHO = '  ' THEN          /*CURSOR SET TO*/
                         CURSOR_VALUE = 339;       /*SEARCH POSITION*/
                    ELSE
                          IF  BDATAO = '  ' |        /*CURSOR SET TO*/
                           (BACTIONO = 'D' |          /*DATA POSITION*/
                            BACTIONO = 'U' |
                            BACTIONO = 'A' |
                            BACTIONO = 'E' ) THEN
                                 CURSOR_VALUE = 419;

                                                      /*SEND INPUT MAP */
          IF CURSOR_VALUE = 0 THEN
            EXEC CICS SEND MAP('DSN8CPE') MAPSET('DSN8CPE');
          ELSE
            EXEC CICS SEND MAP('DSN8CPE') MAPSET('DSN8CPE') ERASE
                  CURSOR(CURSOR_VALUE);


          IF EXITCODE = '1' THEN                       /*FINISHED ? */
            EXEC CICS RETURN; /* RETURN, DON'T REINVOKE TRANSACTION */
          ELSE
            EXEC CICS RETURN TRANSID('D8PP'); /* STANDARD RETURN    */
       END;

0                    /* SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME */
     ELSE EXEC CICS ABEND ABCODE ('MAPO');
  END;
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CP7

THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS THE PARAMETER AREA.

```
 DSN8CP7:PROC (COMMPTR) OPTIONS(MAIN);
 /*****************************************************************
 *                                                                *
 *   MODULE NAME = DSN8CP7                                        *
 *                                                                *
```

```
*   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                        *
*                       SQL 1 MAINLINE                                *
*                       CICS                                          *
*                       PL/I                                          *
*                       PROJECT APPLICATION                           *
*                                                                     *
*    COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985           *
*    REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083            *
*                                                                     *
*    STATUS = RELEASE 2, LEVEL 0                                      *
*                                                                     *
*   FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE      *
*              SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS        *
*              THE PARAMETER AREA.                                    *
*              INCLUDE DSN8MP1.                                       *
*              CALL DSN8CP8.                                          *
*              RETURN TO DSN8CP6.                                     *
*                                                                     *
*   NOTES =                                                           *
*      DEPENDENCIES = CALLED BY DSN8CP6, CALLS DSN8CP8 (CICS LINKS). *
*      RESTRICTIONS = NONE                                            *
*                                                                     *
*                                                                     *
*   MODULE TYPE = PL/I PROC(COMMPTR) OPTIONS.                         *
*      PROCESSOR = DB2  PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER *
*      MODULE SIZE = SEE LINK-EDIT                                    *
*      ATTRIBUTES = REUSABLE                                          *
*                                                                     *
*   ENTRY POINT = DSN8CP7                                             *
*      PURPOSE = SEE FUNCTION                                         *
*      LINKAGE = NONE                                                 *
*                                                                     *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:         *
*                                                                     *
*              SYMBOLIC LABEL/NAME = COMMPTR (POINTER TO COMMAREA)    *
*              DESCRIPTION = NONE                                     *
*                                                                     *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                       *
*                                                                     *
*              SYMBOLIC LABEL/NAME = NONE                             *
*              DESCRIPTION = NONE                                     *
*                                                                     *
*   EXIT-NORMAL = DSN8CP6                                             *
*                                                                     *
*   EXIT-ERROR = DSN8CP6                                              *
*                                                                     *
*      RETURN CODE = NONE                                             *
*                                                                     *
*      ABEND CODES = NONE                                             *
*                                                                     *
*      ERROR-MESSAGES =  NONE                                         *
*                                                                     *
*   EXTERNAL REFERENCES =                                             *
*      ROUTINES/SERVICES =   DSN8CP8                                  *
*                                                                     *
*      DATA-AREAS =                                                   *
*         DSN8MPCA            - PLI STRUCTURE FOR COMMAREA            *
*         DSN8MPCS            - DECLARE CONVERSATION STATUS           *
*         DSN8MPOV            - DECLARE OPTION VALIDATION             *
*         DSN8MPVO            - FIND VALID OPTIONS FOR ACTION,        *
*                               OBJECT, SEARCH CRITERIA               *
*         DSN8MP1             - RETRIEVE LAST CONVERSATION,           *
*                               VALIDATE, CALL SQL2                   *
*         DSN8MP3 -- DSN8MP5  - VALIDATION MODULES CALLED BY DSN8MP1 *
*         DSN8MPXX            - SQL ERROR HANDLER                     *
*                                                                     *
*      CONTROL-BLOCKS =                                               *
*         SQLCA               - SQL COMMUNICATION AREA                *
*                                                                     *
*   TABLES =  NONE                                                    *
*                                                                     *
*   CHANGE-ACTIVITY =  NONE                                           *
*                                                                     *
*                                                                     *
*  *PSEUDOCODE*                                                       *
*                                                                     *
*    PROCEDURE                                                        *
*      INCLUDE DECLARATIONS.                                          *
*      INCLUDE DSN8MP1.                                               *
*      INCLUDE ERROR HANDLER.                                         *
*                                                                     *
*      CP1EXIT:  ( REFERENCED BY DSN8MP1 )                            *
```

```
*          EXEC CICS RETURN.                                        *
*                                                                   *
*      CP1CALL:  ( REFERENCED BY DSN8MP1 )                          *
*          EXEC CICS LINK PROGRAM('DSN8CP8') COMMAREA(COMMAREA)     *
*                                            LENGTH(3000).          *
*          GO TO MP1SAVE.  (LABEL IN DSN8MP1)                       *
*                                                                   *
*      INCLUDE VALIDATION MODULES.                                  *
*                                                                   *
*    END.                                                           *
*********************************************************************/
/*-----------------------------------------------------------------*/
/*                                                                 */
/*                                                                 */
/*                                                                 */
/*                                                                 */
/*                        SQL1   MAINLINE                          */
/*                                                                 */
/*                                                                 */
/*                                                                 */
/*                                                                 */
/*                                                                 */
/*-----------------------------------------------------------------*/
                                        /* SQL RETURN CODE HANDLING */
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

    /***************************************************************/
    /*       ** DCLGENS AND INITIALIZATIONS                        */
    /***************************************************************/

DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
                                        /* DECLARE CONTROL FLAGS */
DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

    /*****************************************************/
    /*          **  FIELDS SENT TO MESSAGE ROUTINE         */
    /*****************************************************/

DCL  MODULE            CHAR (07);
DCL  OUTMSG            CHAR (69);

DCL DSN8MPG EXTERNAL ENTRY;
EXEC SQL INCLUDE DSN8MPCA;                 /* INCLUDE COMMAREA */
DSN8_MODULE_NAME.MAJOR = 'DSN8CP7 ';       /* INITIALIZE MODULE NAME*/
EXEC SQL INCLUDE DSN8MPCS;                 /* INCLUDE PCONA    */
EXEC SQL INCLUDE DSN8MPOV;                 /* INCLUDE POPTVAL  */
EXEC SQL INCLUDE DSN8MPVO;                 /* INCLUDE CURSOR   */
EXEC SQL INCLUDE SQLCA;                    /* INCLUDE SQL COMMAREA*/
EXEC SQL INCLUDE DSN8MP1;                  /* INCLUDE SQL1 MAIN*/
EXEC SQL INCLUDE DSN8MPXX;                 /* INCLUDE ERRORHANDLER */

CP1EXIT :
     EXEC CICS RETURN;                     /*  STANDARD EXIT   */

CP1CALL :
                                      /* GO TO DSN8CP8 (SQL2)     */
   EXEC CICS LINK PROGRAM('DSN8CP8') COMMAREA(COMMAREA) LENGTH(3000);
   GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3;              /* INCLUDE ACTION VALIDATION*/
EXEC SQL INCLUDE DSN8MP4;              /* INCLUDE OBJECT VALIDATION*/
EXEC SQL INCLUDE DSN8MP5;              /* INCLUDE SEARCH CRITERIA*/
END;                                   /* VALIDATION             */
```

Related reference

"Sample applications in CICS" on page 1378

A set of Db2 sample applications run in the CICS environment.

DSN8CP8

ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN CALLS SECONDARY SELECTION MODULES DSN8MPM CALLS DETAIL MODULES DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ CALLED BY DSN8CP7 (SQL1) .

```
   DSN8CP8: PROC(COMMPTR) OPTIONS(MAIN);                                 00010000
                                                                %PAGE;   00020000
 /********************************************************************* 00030000
 *                                                                     * 00040000
 *   MODULE NAME = DSN8CP8                                             * 00050000
 *                                                                     * 00060000
 *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                        * 00070000
 *                      SQL 2 COMMON MODULE                            * 00080000
 *                      CICS                                           * 00090000
 *                      PL/I                                           * 00100000
 *                      PROJECT APPLICATION                            * 00110000
 *                                                                     * 00120000
 *                                                                     * 00130000
 *     LICENSED MATERIALS - PROPERTY OF IBM                            * 00140000
 *     5695-DB2                                                        * 00146000
 *     (C) COPYRIGHT 1982, 1995 IBM CORP.  ALL RIGHTS RESERVED.        * 00153000
 *                                                                     * 00160000
 *     STATUS = VERSION 4                                              * 00170000
 *                                                                     * 00180000
 *   FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN* 00190000
 *              CALLS SECONDARY SELECTION MODULES                      * 00200000
 *                    DSN8MPM                                          * 00210000
 *              CALLS DETAIL MODULES                                   * 00220000
 *                    DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ          * 00230000
 *              CALLED BY DSN8CP7 (SQL1)                               * 00240000
 *                                                                     * 00250000
 *   NOTES = NONE                                                      * 00260000
 *                                                                     * 00270000
 *   MODULE TYPE = BLOCK OF PL/I CODE                                  * 00280000
 *      PROCESSOR   = DB2  PRECOMPILER, PL/I OPTIMIZER                 * 00290000
 *      MODULE SIZE = SEE LINKEDIT                                     * 00300000
 *      ATTRIBUTES  = REUSABLE                                         * 00310000
 *                                                                     * 00320000
 *   ENTRY POINT = DSN8CP8                                             * 00330000
 *      PURPOSE = SEE FUNCTION                                         * 00340000
 *      LINKAGE = NONE                                                 * 00350000
 *      INPUT   =                                                      * 00360000
 *                                                                     * 00370000
 *                SYMBOLIC LABEL/NAME = COMMPTR                        * 00380000
 *                DESCRIPTION         = POINTER TO COMMAREA            * 00390000
 *                                      (COMMUNICATION AREA)           * 00400000
 *                                                                     * 00410000
 *      OUTPUT  =                                                      * 00420000
 *                                                                     * 00430000
 *                SYMBOLIC LABEL/NAME = COMMPTR                        * 00440000
 *                DESCRIPTION         = POINTER TO COMMAREA            * 00450000
 *                                      (COMMUNICATION AREA)           * 00460000
 *                                                                     * 00470000
 *                                                                     * 00480000
 *   EXIT-NORMAL =                                                     * 00490000
 *                                                                     * 00500000
 *   EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION    * 00510000
 *                SPECIFIED IN DSN8CP8 WILL BE RAISED AND PROGRAM      * 00520000
 *                WILL GO TO THE LABEL DB_ERROR.                       * 00530000
 *                                                                     * 00540000
 *                                                                     * 00550000
 *      RETURN CODE = NONE                                             * 00560000
 *                                                                     * 00570000
 *      ABEND CODES = NONE                                             * 00580000
 *                                                                     * 00590000
 *      ERROR-MESSAGES =                                               * 00600000
 *              DSN8062E-AN OBJECT WAS NOT SELECTED                    * 00610000
 *              DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR                * 00630000
 *              DSN8072E-INVALID SELECTION ON SECONDARY SCREEN         * 00640000
 *                                                                     * 00650000
 *   EXTERNAL REFERENCES = NONE                                        * 00660000
 *      ROUTINES/SERVICES = 6 MODULES LISTED ABOVE                     * 00670000
 *                  DSN8MPG     -  ERROR MESSAGE ROUTINE               * 00680000
 *                                                                     * 00690000
 *      DATA-AREAS =                                                   * 00700000
 *                  DSN8MPAC    -  DCLGEN FOR ACTIVITY TYPES           * 00710000
```

```
*                  DSN8MPAS    -  CURSOR SECONDARY SELECTION FOR     * 00720000
*                                 STAFF                              * 00730000
*                  DSN8MPCA    -  COMMUNICATION AREA BETWEEN MODULES * 00740000
*                  DSN8MPDH    -  CURSOR FOR DISPLAY TEXT FROM       * 00750000
*                                 TDSPTXT TABLE                      * 00760000
*                  DSN8MPDP    -  DCLGEN FOR DEPARTMENT              * 00770000
*                  DSN8MPDT    -  DCLGEN FOR DISPLAY TEXT TABLE      * 00780000
*                  DSN8MPEM    -  DCLGEN FOR EMPLOYEE                * 00790000
*                  DSN8MPEP    -  DCLGEN FOR PROJECT/STAFFING        * 00800000
*                  DSN8MPES    -  CURSOR SECONDARY SELECTION FOR     * 00810000
*                                 ESTIMATES                          * 00820000
*                  DSN8MPOV    -  DCLGEN FOR OPTION VALIDATION       * 00830000
*                  DSN8MPPA    -  DCLGEN FOR PROJECT/ACTIVITIES      * 00840000
*                  DSN8MPPD    -  DCLGEN FOR PROJ STRUCTURE DETAIL   * 00850000
*                  DSN8MPP2    -  DCLGEN FOR PROJ STRUCTURE DETAIL   * 00855000
*                  DSN8MPPE    -  CURSOR PROJECT DETAIL              * 00860000
*                  DSN8MPPJ    -  DCLGEN FOR PROJECTS                * 00870000
*                  DSN8MPPL    -  CURSOR PROJECT LIST                * 00880000
*                  DSN8MPPR    -  DCLGEN FOR PROJ/RESP EMPLOYEE      * 00890000
*                  DSN8MPSA    -  DCLGEN FOR PROJ ACTIVITY LISTING   * 00910000
*                  DSN8MPSL    -  CURSOR STAFFING LIST               * 00920000
*                  DSN8MPS2    -  DCLGEN FOR PROJ ACTIVITY LISTING   * 00930000
*                  DSN8MPFP    -  DCLGEN FOR PROJECT-EMPLOYEE        * 00935000
*                  DSN8MPED    -  DCLGEN FOR EMPLOYEE-DEPT           * 00937000
*                  DSN8MPM     -  SECONDARY SELECTION FOR PROJECTS   * 00940000
*                  DSN8MPT     -  PROJECT ACTIVITY LIST              * 00950000
*                  DSN8MPV     -  PROJECT STRUCTURE DETAIL           * 00960000
*                  DSN8MPW     -  ACTIVITY STAFFING DETAIL           * 00970000
*                  DSN8MPX     -  ACTIVITY ESTIMATE DETAIL           * 00980000
*                  DSN8MPZ     -  PROJECT DETAIL                     * 00990000
*                                                                    * 01000000
*      CONTROL-BLOCKS =                                              * 01010000
*         SQLCA                - SQL COMMUNICATION AREA              * 01020000
*                                                                    * 01030000
*   TABLES =  NONE                                                   * 01040000
*                                                                    * 01050000
*   CHANGE-ACTIVITY =  NONE                                          * 01060000
*                                                                    * 01070000
*                                                                    * 01080000
*  *PSEUDOCODE*                                                      * 01090000
*                                                                    * 01100000
*      THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR       * 01110000
*      DETAIL MODULE(S) ARE TO BE CALLED IN THE CICS/PL/I            * 01120000
*      ENVIRONMENT.                                                  * 01130000
*                                                                    * 01140000
*      WHAT HAS HAPPENED SO FAR?.............THE SUBSYSTEM           * 01150000
*      DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS                * 01160000
*      READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL * 01170000
*      TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT   * 01180000
*      FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF    * 01190000
*      ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS      * 01200000
*      MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH     * 01210000
*      POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE    * 01220000
*      BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION      * 01230000
*      AND DETAIL MODULES.                                           * 01240000
*                                                                    * 01250000
*     WHAT IS INCLUDED IN THIS MODULE?.............                  * 01260000
*      ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.    * 01270000
*      ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE        * 01280000
*      SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND                * 01290000
*      SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE     * 01300000
*      RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE* 01310000
*      THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED  * 01320000
*      IN THIS PROCEDURE.                                            * 01330000
*                                                                    * 01340000
*    PROCEDURE                                                       * 01350000
*         IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR              * 01360000
*         IS NOT WILLING TO ACCEPT AN ANSWER THEN                    * 01370000
*                         NEW REQUEST*                               * 01380000
*                                                                    * 01390000
*         ELSE                                                       * 01400000
*                IF ANSWER TO A SECONDARY SELECTION THEN             * 01410000
*                         DETERMINE IF NEW REQUEST.                  * 01420000
*                                                                    * 01430000
*                                                                    * 01440000
*                                                                    * 01450000
*         CASE (NEW REQUEST)                                         * 01460000
*                                                                    * 01470000
*                SUBCASE ('ADD')                                     * 01480000
*                        DETAIL PROCESSOR                            * 01490000
*                        RETURN TO SQL 1                             * 01500000
*                ENDSUB                                              * 01510000
```

```
 *                                                                    * 01520000
 *                SUBCASE ('DISPLAY','ERASE','UPDATE')                * 01530000
 *                        CALL SECONDARY SELECTION                    * 01540000
 *                        IF # OF POSSIBLE CHOICES IS ^= 1 THEN       * 01550000
 *                               RETURN TO SQL 1                      * 01560000
 *                        ELSE                                        * 01570000
 *                               CALL THE DETAIL PROCESSOR            * 01580000
 *                               RETURN TO SQL 1                      * 01590000
 *                ENDSUB                                              * 01600000
 *                                                                    * 01610000
 *         ENDCASE                                                    * 01620000
 *                                                                    * 01630000
 *         IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS       * 01640000
 *         ACTUALLY BEEN MADE THEN                                    * 01650000
 *                     VALID SELECTION #?                             * 01660000
 *                     IF IT IS VALID THEN                            * 01670000
 *                           CALL DETAIL PROCESSOR                    * 01680000
 *                           RETURN TO SQL 1                          * 01690000
 *                     ELSE                                           * 01700000
 *                           PRINT ERROR MSG                          * 01710000
 *                           RETURN TO SQL 1.                         * 01720000
 *                                                                    * 01730000
 *         IF ANSWER TO SECONDARY SELECTION THEN                      * 01740000
 *                  CALL SECONDARY SELECTION                          * 01750000
 *                  RETURN TO SQL 1.                                  * 01760000
 *                                                                    * 01770000
 *         IF ANSWER TO DETAIL THEN                                   * 01780000
 *                  CALL DETAIL PROCESSOR                             * 01790000
 *                  RETURN TO SQL 1.                                  * 01800000
 *                                                                    * 01810000
 *     END.                                                           * 01820000
 *                                                                    * 01830000
 *        *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES* 01840000
 *         THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.        * 01850000
 *                                                                    * 01860000
 *-------------------------------------------------------------------*/ 01870000
                                                                        01880000
   EXEC SQL INCLUDE DSN8MPCA;   /*COMMUNICATION AREA BETWEEN MODULES */ 01890000
   EXEC SQL INCLUDE SQLCA;      /*SQL COMMUNICATION AREA             */ 01900000
                                                                        01910000
   EXEC SQL INCLUDE DSN8MPDP;   /* DCLGEN FOR DEPARTMENT             */ 01920000
   EXEC SQL INCLUDE DSN8MPEM;   /* DCLGEN FOR EMPLOYEE               */ 01930000
   EXEC SQL INCLUDE DSN8MPPJ;   /* DCLGEN FOR PROJECTS               */ 01940000
   EXEC SQL INCLUDE DSN8MPAC;   /* DCLGEN FOR ACTIVITY TYPES         */ 01950000
   EXEC SQL INCLUDE DSN8MPPA;   /* DCLGEN FOR PROJECT/ACTIVITIES     */ 01960000
   EXEC SQL INCLUDE DSN8MPEP;   /* DCLGEN FOR PROJECT/STAFFING       */ 01970000
   EXEC SQL INCLUDE DSN8MPPR;   /* DCLGEN FOR PROJ/RESP EMPLOYEE     */ 01980000
   EXEC SQL INCLUDE DSN8MPPD;   /* DCLGEN FOR PROJ STRUCTURE DETAIL  */ 01990000
   EXEC SQL INCLUDE DSN8MPP2;   /* DCLGEN FOR PROJ STRUCTURE DETAIL  */ 02000000
   EXEC SQL INCLUDE DSN8MPSA;   /* DCLGEN FOR PROJ ACTIVITY LISTING  */ 02010000
   EXEC SQL INCLUDE DSN8MPS2;   /* DCLGEN FOR PROJ ACTIVITY LISTING  */ 02020000
   EXEC SQL INCLUDE DSN8MPFP;   /* DCLGEN FOR PROJECT-EMPLOYEE       */ 02025000
   EXEC SQL INCLUDE DSN8MPED;   /* DCLGEN FOR EMPLOYEE-DEPT          */ 02027000
                               /* PROGRAMMING TABLES                 */ 02030000
   EXEC SQL INCLUDE DSN8MPOV;   /* DCLGEN FOR OPTION VALIDATION      */ 02040000
   EXEC SQL INCLUDE DSN8MPDT;   /* DCLGEN FOR DISPLAY TEXT TABLE     */ 02050000
                                                                        02060000
    /* CURSORS */                                                       02070000
   EXEC SQL INCLUDE DSN8MPPL;    /* MAJSYS P - SEC SEL FOR PS, AL, PR*/ 02080000
   EXEC SQL INCLUDE DSN8MPES;    /* MAJSYS P - SEC SEL FOR AE        */ 02090000
   EXEC SQL INCLUDE DSN8MPAS;    /* MAJSYS P - SEC SEL FOR AS        */ 02100000
   EXEC SQL INCLUDE DSN8MPPE;    /* MAJSYS P - DETAIL  FOR PS        */ 02110000
   EXEC SQL INCLUDE DSN8MPSL;    /* MAJSYS P - DETAIL  FOR AL        */ 02120000
   EXEC SQL INCLUDE DSN8MPDH;    /* PROG TABLES - DISPLAY HEADINGS   */ 02130000
                                                                        02140000
  /*********************************************************/          02150000
  /* SQL RETURN CODE HANDLING                               */          02160000
  /*********************************************************/          02170000
                                                                        02180000
   EXEC SQL WHENEVER SQLERROR   GO TO DB_ERROR;                         02190000
   EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;                         02200000
                                                                        02210000
0  DCL UNSPEC BUILTIN;                                                  02220000
   DCL VERIFY BUILTIN;                                                  02230000
                                                                        02240000
   DCL LENGTH BUILTIN;                                                  02250000
   DCL DSN8MPG EXTERNAL ENTRY;                                          02260000
                                                                        02270000
     /***************************************************************/ 02280000
     /*       ** DCLGENS AND INITIALIZATIONS                        */ 02290000
     /***************************************************************/ 02300000
                                                                        02310000
```

```
DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
                                        /* DECLARE CONTROL FLAGS */
DCL ( SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);

 /*********************************************/
 /* FIELDS SENT TO MESSAGE ROUTINE          */
 /*********************************************/

   DCL MODULE        CHAR (07) INIT('DSN8CP8');
   DCL OUTMSG        CHAR (69);

 /**********************************************************/
 /*   INITIALIZATIONS                                      */
 /**********************************************************/

  DSN8_MODULE_NAME.MAJOR='DSN8CP8';
  DSN8_MODULE_NAME.MINOR=' ';

    /************************************************************/
    /*  DETERMINES WHETHER NEW REQUEST OR NOT                   */
    /************************************************************/

 /* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */
 /* NEW REQUEST.                                                 */

  IF PCONVSTA.PREV = ' ' THEN
    COMPARM.NEWREQ = 'Y';

 /* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A  */
 /* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/
 /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */

  IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' &
     SUBSTR(COMPARM.DATA,3,58) ^= ' ' &
     COMPARM.DATA ^= 'NEXT'
     THEN COMPARM.NEWREQ = 'Y';

    /************************************************************/
    /*  IF NEW REQUEST AND ACTION IS 'ADD' THEN                 */
    /*       CALL DETAIL PROCESSOR                              */
    /*  ELSE CALL SECONDARY SELECTION                           */
    /************************************************************/
  IF COMPARM.NEWREQ='Y' THEN
    DO;

      IF COMPARM.ACTION = 'A' THEN
        DO;
          CALL DETAIL;                      /* CALL DETAIL PROCESSOR*/
          GO TO EXIT;                       /* RETURN */
        END;

      CALL SECSEL;                 /* CALL SECONDARY SELECTION */

      IF MAXSEL = 1 THEN           /* IF NO. OF CHOICES = 1    */
        CALL DETAIL;               /* CALL DETAIL PROCESSOR    */
      GO TO EXIT;                  /* RETURN                   */
    END;

 /* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */
 /* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS                */
 /* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2     */
 /* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION.            */

    /************************************************************/
    /*  DETERMINES IF VALID SELECTION NUMBER                    */
    /************************************************************/

  IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */

  IF PCONVSTA.MAXSEL < 1  THEN GO TO IP201; /* NO VALID CHOICES */

  IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROOL REQUEST*/

  IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2)
                 THEN GO TO IP201;  /* NO CHANGE ON INPUT SCREEN */

  IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN     /* SECOND CHAR BLANK */
     IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN
       DO;
         SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1);
```

```
         SUBSTR(COMPARM.DATA,1,1) = '0';
       END;

  IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 &
            SUBSTR(COMPARM.DATA,1,2) > '00' THEN
        IF DATAP <= PCONVSTA.MAXSEL THEN
          DO;

            COMPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */
            CALL DETAIL;          /* CALL DETAIL PROCESSOR         */
            GO TO EXIT;           /*      RETURN                   */
          END;

                                    /*    INVALID SELECTION NO.    */
       CALL DSN8MPG (MODULE, '072E', OUTMSG); /* PRINT ERROR MSG   */
       PCONVSTA.MSG = OUTMSG;
       PCONVSTA.PREV = ' ';         /* NOT SEC SELECTION, ERROR    */

  GO TO EXIT;                       /*    RETURN                   */

    /*************************************************************/
    /*  DETERMINES WHETHER SECONDARY SELECTION OR DETAIL       */
    /*************************************************************/
 /* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */
 IP201:
  IF PCONVSTA.PREV = 'S' THEN
    DO;
      CALL SECSEL;                     /* CALL SECONDARY SELECTION*/
      GO TO EXIT;                      /* RETURN */
    END;

  IF PCONVSTA.PREV = 'D' THEN
    DO;
      CALL DETAIL;                     /* CALL DETAIL PROCESSOR */
      GO TO EXIT;                      /* RETURN */
    END;

                                     /*  LOGIC ERROR */
                                     /* PRINT ERROR MESSAGE */
  CALL DSN8MPG (MODULE, '066E', OUTMSG);
  PCONVSTA.MSG= OUTMSG;
  PCONVSTA.PREV = ' ';               /* NOT SEC SELECTION, ERROR */
  GO TO EXIT;                        /* RETURN */

   EXEC SQL INCLUDE DSN8MPXX;        /* HANDLES SQL ERRORS */
   GO TO EXIT;                       /* RETURN  */

    /**************************************************************/
    /*  CALLS SECONDARY SELECTION AND RETURNS TO SQL 1           */
    /* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
    /**************************************************************/

SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE */
      PCONVSTA.LASTSCR = 'DSN8001';  /* SET FOR GENERAL MAP  */

      IF COMPARM.OBJFLD='AE' |      /*ACTIVITY ESTIMATE */
         COMPARM.OBJFLD='AL' |      /*PROJECT ACTIVITY LISTING */
         COMPARM.OBJFLD='AS' |      /*INDIVIDUAL PROJECT STAFFING*/
         COMPARM.OBJFLD='PR' |      /*INDIVIDUAL PROJECT PROCESSING*/
         COMPARM.OBJFLD='PS' THEN   /*PROJECT STRUCTURE */
        DO;
          CALL DSN8MPM;           /*SECONDARY SELECTION FOR PROJECTS*/
          RETURN;                 /*RETURN */
        END;

                                            /*MISSING SECONDARY SEL*/
                                            /*PRINT ERROR MESSAGE  */
  CALL DSN8MPG (MODULE, '062E', OUTMSG);
  PCONVSTA.MSG= OUTMSG;
  PCONVSTA.PREV = ' ';               /* NOT SEC SELECTION, ERROR */
  GO TO EXIT;                               /*RETURN */
  END SECSEL;

     /*************************************************************/
     /* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1            */
     /*************************************************************/

DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */
      PCONVSTA.LASTSCR = 'DSN8002';  /* SET FOR DETAIL  MAP  */

      IF COMPARM.OBJFLD='PS' THEN
        DO;
```

```
          CALL DSN8MPV;               /* PROJECT STRUCTURE DETAIL */   03960000
          RETURN;                                                      03970000
        END;                                                           03980000
                                                                       03990000
      IF COMPARM.OBJFLD='AL' THEN                                      04000002
        DO;                                                            04010000
          CALL DSN8MPT;               /* PROJECT ACTIVITY LIST */      04020000
          RETURN;                                                      04030000
        END;                                                           04040000
                                                                       04050000
      IF COMPARM.OBJFLD='PR' THEN                                      04060002
        DO;                                                            04070000
          CALL DSN8MPZ;               /* PROJECT DETAIL */             04080000
          RETURN;                                                      04090000
        END;                                                           04100000
                                                                       04110000
      IF COMPARM.OBJFLD='AE' THEN                                      04120002
        DO;                                                            04130000
          CALL DSN8MPX;                /* ACTIVITY ESTIMATE DETAIL */  04140000
          RETURN;                                                      04150000
        END;                                                           04160000
                                                                       04170000
      IF COMPARM.OBJFLD='AS' THEN                                      04180002
        DO;                                                            04190000
          CALL DSN8MPW;                /* ACTIVITY STAFFING DETAIL */  04200000
          RETURN;                                                      04210000
        END;                                                           04220000
                                                                       04230000
                                        /*MISSING DETAIL MODULE*/      04240000
                                        /*PRINT ERROR MESSAGE  */      04250000
    CALL DSN8MPG (MODULE, '062E', OUTMSG);                             04260000
    PCONVSTA.MSG= OUTMSG;                                              04270000
    PCONVSTA.PREV = ' ';              /* NOT SEC SELECTION, ERROR */   04280000
    GO TO EXIT;                                                        04290000
    END DETAIL;                                                        04300000
                                                                       04310000
                                        /*RETURNS TO SQL 1*/           04320000
   EXIT: EXEC CICS RETURN;                                             04330000
                            /* PROJECTS */                             04340000
         EXEC SQL INCLUDE DSN8MPM;       /* SEC SEL - PROJECTS        */ 04350000
         EXEC SQL INCLUDE DSN8MPT;       /* DETAIL  - PROJ ACT LISTING*/ 04360000
         EXEC SQL INCLUDE DSN8MPV;       /* DETAIL  - PROJ STRUCTURE */  04370000
         EXEC SQL INCLUDE DSN8MPW;       /* DETAIL  - INDIVID STAFFING*/ 04380000
         EXEC SQL INCLUDE DSN8MPX;       /* DETAIL  - INDIVID ACTIVITY*/ 04390000
         EXEC SQL INCLUDE DSN8MPZ;       /* DETAIL  - INDIVIDUAL PROJ */ 04400000
    END DSN8CP8;                                                       04410000
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSN8CP3

THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND UPDATES THEM IF DESIRED.

```
  DSN8CP3: PROC OPTIONS (MAIN);
  /*********************************************************************
  *                                                                    *
  *   MODULE NAME = DSN8CP3                                            *
  *                                                                    *
  *   DESCRIPTIVE NAME = DB2  SAMPLE APPLICATION                       *
  *                       PHONE APPLICATION                            *
  *                       CICS                                         *
  *                       PL/I                                         *
  *                                                                    *
  *     Licensed Materials - Property of IBM                           *
  *     5635-DB2                                                       *
  *     (C) COPYRIGHT 1982, 2006 IBM Corp.  All Rights Reserved.       *
  *                                                                    *
  *     STATUS = Version 9                                             *
  *                                                                    *
  *   FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND          *
  *              UPDATES THEM IF DESIRED.                              *
  *                                                                    *
  *   NOTES =                                                          *
  *      DEPENDENCIES = THREE CICS MAPS(DSECTS) ARE REQUIRED:          *
  *                     DSN8MPMN, DSN8MPML, AND DSN8MPMU               *
  *      RESTRICTIONS = NONE                                           *
```

```
*                                                                   *
*   MODULE TYPE = PL/I PROC OPTIONS(MAIN)                           *
*      PROCESSOR   = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER*
*      MODULE SIZE = SEE LINKEDIT                                   *
*      ATTRIBUTES  = REENTRANT                                      *
*                                                                   *
*   ENTRY POINT =  DSN8CP3                                          *
*      PURPOSE = SEE FUNCTION                                       *
*      LINKAGE = INVOKED FROM CICS                                  *
*                                                                   *
*      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:       *
*              INPUT-MESSAGE:                                       *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPNI                *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)           *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPLI                *
*                     DESCRIPTION = PHONE MENU 2 (LIST)             *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPUI                *
*                     DESCRIPTION = PHONE MENU 3 (UPDATE)           *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = VPHONE                  *
*                     DESCRIPTION = VIEW OF TELEPHONE INFORMATION   *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = VEMPLP                  *
*                     DESCRIPTION = VIEW OF EMPLOYEE INFORMATION    *
*                                                                   *
*      OUTPUT = PARAMETERS EXPLICITLY RETURNED:                     *
*               OUTPUT-MESSAGE:                                     *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPNO                *
*                     DESCRIPTION = PHONE MENU 1 (SELECT)           *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPLO                *
*                     DESCRIPTION = PHONE MENU 2 (LIST)             *
*                                                                   *
*                     SYMBOLIC LABEL/NAME = DSN8CPUO                *
*                     DESCRIPTION = PHONE MENU 3 (UPDATE)           *
*                                                                   *
*   EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION                   *
*                                                                   *
*   EXIT-ERROR =                                                    *
*                                                                   *
*      RETURN CODE = NONE                                           *
*                                                                   *
*      ABEND CODES =  NONE                                          *
*                                                                   *
*                                                                   *
*      ERROR-MESSAGES =                                             *
*            DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED               *
*            DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE    *
*            DSN8008I - NO EMPLOYEE FOUND IN TABLE                  *
*            DSN8057I - FURTHER ENTRIES IN TABLE - UPDATE POSSIBLE  *
*            DSN8060E - SQL ERROR, RETURN CODE IS:                  *
*                                                                   *
*   EXTERNAL REFERENCES =                                           *
*      ROUTINES/SERVICES =                                          *
*         DSN8MPG              - ERROR MESSAGE ROUTINE              *
*                                                                   *
*      DATA-AREAS =                                                 *
*         IN_MESSAGE           - VIA BMS, SEE INPUT PARAMETERS      *
*         OUT_MESSAGE          - VIA BMS, SEE OUTPUT PARAMTERS      *
*         DSN8MPML             - DECLARE FOR DSN8CPL CICS MAP       *
*         DSN8MPMN             - DECLARE FOR DSN8CPN CICS MAP       *
*         DSN8MPMU             - DECLARE FOR DSN8CPU CICS MAP       *
*                                                                   *
*      CONTROL-BLOCKS =                                             *
*         SQLCA                - SQL COMMUNICATION AREA             *
*                                                                   *
*   TABLES = NONE                                                   *
*                                                                   *
*                                                                   *
*   CHANGE-ACTIVITY =                                               *
*   PQ92146 09/07/04 CHANGE DECLARED LENGTH OF BMS_IO FROM 32767  @01*
*                    TO 1408 TO STOP IBM2402I COMPILE-TIME ERROR  @01*
*                                                                   *
*                                                                   *
*  *PSEUDOCODE*                                                     *
*                                                                   *
*      PROCEDURE                                                    *
*         GET FIRST INPUT                                           *
```

```
 *         DO WHILE MORE INPUT                                        *
 *            GET REPORT HEADING                                      *
 *                                                                    *
 *            CASE (ACTION)                                           *
 *                                                                    *
 *               SUBCASE ('L')                                        *
 *                 IF LASTNAME IS '*'                                 *
 *                   LIST ALL EMPLOYEES                               *
 *                 ELSE                                               *
 *                   IF LASTNAME CONTAINS '%'                         *
 *                      LIST EMPLOYEES GENERIC                        *
 *                   ELSE                                             *
 *                      LIST EMPLOYEES SPECIFIC                       *
 *               ENDSUB                                               *
 *                                                                    *
 *               SUBCASE ('U')                                        *
 *                   UPDATE PHONENUMBER FOR EMPLOYEE                  *
 *                   WRITE CONFIRMATION MESSAGE                       *
 *               OTHERWISE                                            *
 *                   INVALID REQUEST                                  *
 *               ENDSUB                                               *
 *                                                                    *
 *         GET NEXT INPUT                                             *
 *      ENDCASE                                                       *
 *                                                                    *
 *         IF SQL ERROR OCCURS THEN                                   *
 *           FORMAT ERROR MESSAGE                                     *
 *           ROLLBACK                                                 *
 *         END                                                        *
 * END.                                                               *
 *-------------------------------------------------------------------*/
 /*------------------------------------------------------------------*
 *                                                                    *
 *   MODULE NAME = DSN8CP3                                            *
 *     KDB0010                                                        *
 *                                                                    *
 *                                                                    *
 *-------------------------------------------------------------------*/
1/*******************************************************************/
 /*      DECLARATION FOR INPUT / OUTPUT                              */
 /*******************************************************************/
 EXEC SQL INCLUDE DSN8MPMN ;
 EXEC SQL INCLUDE DSN8MPML ;
 EXEC SQL INCLUDE DSN8MPMU ;
0DCL 1 SUBMAPI(15)  UNALIGNED BASED(ADDR(DSN8CU2I.NEWNO1L)),
     2 NEWNOL FIXED BIN(15,0),
     2 NEWNOA CHAR(1),
     2 NEWNOD CHAR(4),
     2 ENOL FIXED BIN(15,0),
     2 ENOA CHAR(1),
     2 ENOD CHAR(6);
0DCL 1 SUBMAPO(15) UNALIGNED BASED(ADDR(DSN8CL2I.FNAME1L)),
     2 FNAMEL FIXED BIN(15,0),
     2 FNAMEA CHAR(1),
     2 FNAMED CHAR(12),
     2 MINITL FIXED BIN(15,0),
     2 MINITA CHAR(1),
     2 MINITD CHAR(1),
     2 LNAMEL FIXED BIN(15,0),
     2 LNAMEA CHAR(1),
     2 LNAMED CHAR(15),
     2 PNOL FIXED BIN(15,0),
     2 PNOA CHAR(1),
     2 PNOD CHAR(4),
     2 ENOL FIXED BIN(15,0),
     2 ENOA CHAR(1),
     2 ENOD CHAR(6),
     2 WDEPTL FIXED BIN(15,0),
     2 WDEPTA CHAR(1),
     2 WDEPTD CHAR(3),
     2 WNAMEL FIXED BIN(15,0),
     2 WNAMEA CHAR(1),
     2 WNAMED CHAR(31);

 /***** HOLDS BYTE-COUNT OF STORAGE ALLOCATED TO BMS OUTPUT AREA *****/
 DCL  BMS_LL       BIN FIXED(    31 )  INIT(  STG(DSN8CL2I) );/*@EDVG*/

 /******* MASK/OVERLAY OF STORAGE ALLOCATED TO BMS OUTPUT AREA *******/
 DCL  BMS_IO            CHAR( 1408 ) BASED( ADDR(DSN8CL2I) );   /*@01*/

1/*******************************************************************/
 /*       DECLARATION FOR PGM-LOGIC                                  */
```

```
 /*****************************************************************/
 DCL  FIRST        BIT(1);
 DCL  PAGING       BIT(1);
 DCL  OFLOW        BIT(1);
 DCL  EMPLOYEE_NO  CHAR (6);
 DCL  PHONE_NO     CHAR (4);
 DCL  CHAR_SQLCODE CHAR (14);
 DCL 1 CHAR_SQLSTR BASED(ADDR(CHAR_SQLCODE)),
     2 CHAR_BLNK   CHAR(4),
     2 CHAR_SQLCOD CHAR(10);

1/*************************************************/
 /*  FIELDS SENT TO MESSAGE ROUTINE                */
 /*************************************************/
 DCL  MODULE       CHAR (7)  INIT('DSN8CP3');
 DCL  OUTMSG       CHAR (69);

 DCL  DSN8MPG  EXTERNAL ENTRY;
1/*****************************************************************/
 /*                         DECLARATION FOR SQL                   */
 /*****************************************************************/
0EXEC SQL INCLUDE SQLCA;           /* SQL COMMUNICATION AREA        */
                                   /* SQL DECLARATION FOR VIEW PHONE */
 EXEC SQL DECLARE VPHONE TABLE
        (LASTNAME       VARCHAR(15)         ,
         FIRSTNAME      VARCHAR(12)         ,
         MIDDLEINITIAL     CHAR(1)          ,
         PHONENUMBER       CHAR(4)          ,
         EMPLOYEENUMBER    CHAR(6)          ,
         DEPTNUMBER        CHAR(3)  NOT NULL,
         DEPTNAME       VARCHAR(36) NOT NULL);
                                   /* STUCTURE FOR PHONE RECORD      */
 DCL 1 PPHONE,
       2 LASTNAME         CHAR (15) VAR,
       2 FIRSTNAME        CHAR (12) VAR,
       2 MIDDLEINITIAL    CHAR (1),
       2 PHONENUMBER      CHAR (4),
       2 EMPLOYEENUMBER   CHAR (6),
       2 DEPTNUMBER       CHAR (3),
       2 DEPTNAME         CHAR (36) VAR;
                                   /* SQL DECLARATION FOR VIEW VEMPLP*/
 EXEC SQL DECLARE VEMPLP TABLE
        (EMPLOYEENUMBER    CHAR(6)          ,
         PHONENUMBER       CHAR(4));
                                   /* STRUCTURE FOR PEMPLP RECORD    */
 DCL 1 PEMP,
       2 EMPLOYEENUMBER   CHAR (6),
       2 PHONENUMBER      CHAR (4);

1/*****************************************************************/
 /*               SQL CURSORS                                      */
 /*****************************************************************/

 EXEC SQL DECLARE TELE1 CURSOR FOR
          SELECT *
          FROM VPHONE;

 EXEC SQL DECLARE TELE2 CURSOR FOR
          SELECT *
          FROM  VPHONE
          WHERE LASTNAME LIKE :DSN8CN2I.LNAMEI
            AND FIRSTNAME LIKE :DSN8CN2I.FNAMEI;

 EXEC SQL DECLARE TELE3 CURSOR FOR
          SELECT *
          FROM  VPHONE
          WHERE LASTNAME  =    :DSN8CN2I.LNAMEI
            AND FIRSTNAME LIKE :DSN8CN2I.FNAMEI;

1/*****************************************************************/
 /*               SQL RETURN CODE HANDLING                         */
 /*****************************************************************/

 EXEC SQL WHENEVER SQLERROR   GOTO P3_DBERROR;
 EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;
 EXEC SQL WHENEVER NOT FOUND  CONTINUE;

1/*****************************************************************/
 /*                 MAIN PROGRAM ROUTINE                           */
 /*****************************************************************/
                                           /* SET HANDLE CONDITIONS  */
    EXEC CICS HANDLE CONDITION MAPFAIL (P3_MAPFAIL);
```

```
    EXEC CICS HANDLE AID CLEAR (P3_CLEAR);

 /******************** CLEAR THE BMS OUTPUT AREA ********************/
    SUBSTR(BMS_IO,1,BMS_LL) = LOW(BMS_LL) ;                  /*@EDVG*/

 P3_START:
    FIRST = '1'B;                             /*INITIALIZE FIRST BIT */
    OFLOW = '0'B;                           /*INITIALIZE OVERFLOW BIT*/

    SELECT (EIBTRNID);                       /* SELECT ACTION */
      WHEN ('D8PT') DO;                      /* LIST EMPLOYEES */

                                             /* GET INPUT FROM SCREEN */
        EXEC CICS RECEIVE MAP('DSN8CN2') MAPSET('DSN8CPN');

1/*******************************************************************/
 /*                 LIST ALL EMPLOYEES                              */
 /*******************************************************************/

        IF DSN8CN2I.LNAMEI = '*' THEN      /*LIST ALL EMPLOYEES */
         DO;

          EXEC SQL OPEN TELE1;                /* OPEN CURSOR    */

          EXEC SQL FETCH TELE1             /* GET FIRST RECORD  */
                   INTO :PPHONE;

          I = 0;                          /* INITIALIZE COUNTER */

          IF SQLCODE = 100 THEN            /* NO EMPLOYEE FOUND */
           DO;                           /* PRINT ERROR MESSAGE */
            CALL DSN8MPG (MODULE, '008I', OUTMSG);
            DSN8CN3I.EMSGI = OUTMSG;
            EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
            EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
           END;

           DO WHILE (SQLCODE = 0);                 /*LIST EMPLOYEES*/
             I = I + 1;                         /* INCREMENT COUNTER*/
             PAGING = '1'B;
             SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
             SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
             SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
             SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
             SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
             SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
             SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

             IF I = 15 THEN                  /*POSSIBLE OVERFLOW */
              DO;                            /* PRINT ERROR MESSAGE*/
                OFLOW = '1'B;
                CALL DSN8MPG (MODULE, '057I', OUTMSG);
                DSN8CL3O.EMSGO = OUTMSG;
              END;

             IF I = 15 THEN LEAVE;             /* SCREEN IS FILLED */

             EXEC SQL FETCH TELE1            /* GET NEXT RECORD   */
                     INTO :PPHONE;
           END;                              /* END OF WHILE      */

          EXEC SQL CLOSE TELE1;          /* CLOSE CURSOR         */
         END;                            /* END OF IF            */
1/*******************************************************************/
 /*                 LIST GENERIC EMPLOYEES                          */
 /*******************************************************************/

        ELSE                               /* SELECT EMPLOYEES BY NAME*/
         DO;                               /* SEARCH ON PART OF NAME? */
           IF DSN8CN2I.LNAMEL = 0 THEN     /* IS LAST NAME BLANK?     */
              DSN8CN2I.LNAMEI = '%%%%%%%%%%%%%%%'; /* YES, ANYTHING  */
           IF INDEX(DSN8CN2I.LNAMEI,'%') > 0 THEN  /* IS IT A PATTERN*/
            DO;                             /* YES, SEARCH ON         */
                                           /* PART OF LAST NAME       */
              DSN8CN2I.LNAMEI = TRANSLATE(DSN8CN2I.LNAMEI,'%',' ');

                                           /*AND OPTIONALLY FIRST NAME*/
              IF DSN8CN2I.FNAMEL = 0 THEN
                 DSN8CN2I.FNAMEI = '%%%%%%%%%%%%';
              ELSE
                 DSN8CN2I.FNAMEI = TRANSLATE(DSN8CN2I.FNAMEI,'%',' ');
```

```
               EXEC SQL OPEN TELE2;            /* OPEN CURSOR         */

               EXEC SQL FETCH TELE2        /* GET FIRST RECORD        */
                       INTO :PPHONE;

               I = 0;                      /* INITIALIZE COUNTER      */

               IF SQLCODE = 100 THEN       /* EMPLOYEE NOT FOUND      */
                DO;                        /* PRINT ERROR MESSAGE     */
                  CALL DSN8MPG (MODULE, '008I', OUTMSG);
                  DSN8CN3I.EMSGI = OUTMSG;
                  EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
                  EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
                END;

               DO WHILE (SQLCODE = 0);        /* LIST EMPLOYEES */
                  I = I + 1;                  /* INCREMENT COUNTER */
                  PAGING = '1'B;
                  SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
                  SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
                  SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
                  SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
                  SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
                  SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
                  SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

              IF I = 15 THEN                 /*POSSIBLE OVERFLOW */
               DO;                           /* PRINT ERROR MESSAGE*/
                 OFLOW = '1'B;
                 CALL DSN8MPG (MODULE, '057I', OUTMSG);
                 DSN8CL3O.EMSGO = OUTMSG;
               END;

                  IF I = 15 THEN LEAVE;      /* SCREEN IS FILLED */

                  EXEC SQL FETCH TELE2        /* GET NEXT RECORD  */
                          INTO :PPHONE;
               END;                           /* END OF DO WHILE  */

                  EXEC SQL CLOSE TELE2;       /* CLOSE CURSOR     */
             END;                             /* END OF IF        */

 1/**********************************************************************/
  /*                 LIST SPECIFIC EMPLOYEE(S)                          */
  /**********************************************************************/

            ELSE                            /*  SEARCH ON LAST NAME    */
             DO;                            /*AND OPTIONALLY FIRST NAME*/
               IF DSN8CN2I.FNAMEL = 0 THEN
                 DSN8CN2I.FNAMEI = '%%%%%%%%%%%%';
               ELSE
                 DSN8CN2I.FNAMEI = TRANSLATE(DSN8CN2I.FNAMEI,'%',' ');

               EXEC SQL OPEN TELE3;            /* OPEN CURSOR         */

               EXEC SQL FETCH TELE3            /* GET FIRST RECORD    */
                       INTO :PPHONE;

               I = 0;                        /* INITIALIZE COUNTER   */

               IF SQLCODE = 100 THEN          /* EMPLOYEE NOT FOUND  */
                DO;                           /* PRINT ERROR MESSAGE */
                 CALL DSN8MPG (MODULE, '008I', OUTMSG);
                 DSN8CN3I.EMSGI = OUTMSG;
                 EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
                 EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
                END;

               DO WHILE (SQLCODE = 0);       /* LIST EMPLOYEE(S)  */
                  I = I + 1;                 /* INCREMENT COUNTER */
                  PAGING = '1'B;
                  SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
                  SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
                  SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
                  SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
                  SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
                  SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
                  SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

               IF I = 15 THEN                 /*POSSIBLE OVERFLOW */
                DO;                           /* PRINT ERROR MESSAGE*/
                  OFLOW = '1'B;
```

```
                 CALL DSN8MPG (MODULE, '057I', OUTMSG);
                 DSN8CL3O.EMSGO = OUTMSG;
               END;

                 IF I = 15 THEN LEAVE;     /* SCREEN IS FILLED       */

                 EXEC SQL FETCH TELE3     /*   GET NEXT RECORD       */
                          INTO :PPHONE;
              END;                        /* END OF DO WHILE         */


              EXEC SQL CLOSE TELE3;       /* CLOSE CURSOR        */
            END;                          /* END OF ELSE         */
          END;                            /* END OF IF           */

         IF PAGING THEN
          DO;
           PAGING = '0'B;
           EXEC CICS SEND MAP ('DSN8CL1') MAPSET('DSN8CPL') ERASE
           ACCUM PAGING;
           EXEC CICS SEND MAP ('DSN8CL2') MAPSET('DSN8CPL')
           ACCUM PAGING;

           IF OFLOW THEN
            DO;
              OFLOW = '0'B;
              EXEC CICS SEND MAP ('DSN8CL3') MAPSET('DSN8CPL')
              ACCUM PAGING;
            END;

           EXEC CICS SEND PAGE;
           EXEC CICS RETURN TRANSID('D8PU');
          END;                                    /* END OF IF    */

         ELSE EXEC CICS RETURN TRANSID ('D8PT');
       END;                                       /* END OF WHEN  */
                                     /* CHANGE ERROR HANDLING   */
                                     /* FOR UPDATE              */
       EXEC SQL WHENEVER SQLERROR CONTINUE;
       EXEC SQL WHENEVER SQLWARNING CONTINUE;
 1/*******************************************************************/
  /*             UPDATES PHONE NUMBERS FOR EMPLOYEES                */
  /******************************************************************/
       WHEN ('D8PU') DO;                     /* TELEPHONE UPDATE        */

                                     /* GET UPDATED DATA        */
         EXEC CICS RECEIVE MAP('DSN8CU2') MAPSET('DSN8CPU');
                                     /* FIND WHICH NUMBERS HAVE */
                                     /* BEEN UPDATED            */
         DSN8CN3I.EMSGI = '';              /* SET IN CASE NO UPDATES  */

         DO I = 1 TO 15;
           IF SUBMAPI.NEWNOL(I) = 0 THEN;  /* NO UPDATE ON THIS LINE  */
           ELSE
            DO;
             EMPLOYEE_NO = SUBMAPI.ENOD(I);
             PHONE_NO    = SUBMAPI.NEWNOD(I);

             EXEC SQL UPDATE VEMPLP         /*     PERFORM UPDATE     */
                      SET PHONENUMBER = :PHONE_NO
                      WHERE EMPLOYEENUMBER = :EMPLOYEE_NO;

             IF SQLCODE ^= 0 THEN
              DO;                          /* UPDATE FAILED */
                                           /* PRINT ERROR MESSAGE */
               CALL DSN8MPG (MODULE, '007E', OUTMSG);
               DSN8CU3I.EMSGI = OUTMSG;
               EXEC CICS SEND MAP('DSN8CU3') MAPSET('DSN8CPU');
               GOTO P3_DBERROR2;
              END;
                                             /* UPDATE SUCCESSFUL*/
                                           /* PRINT CONFIRMATION */
             ELSE                          /* MESSAGE            */
              DO;
               CALL DSN8MPG (MODULE, '004I', OUTMSG);
               DSN8CN3I.EMSGI = OUTMSG;
              END;
            END;                              /* END ELSE            */
         END;                                /* END FOR             */

         EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
         EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN') ;
```

```
        EXEC CICS RETURN TRANSID('D8PT');
      END;                               /* END WHEN               */
      OTHERWISE GOTO P3_CLEAR;          /* WRONG TX CODE          */
    END;                                /* END SELECT             */
    GOTO P3_END;
 P3_MAPFAIL:                            /* D8PT FROM UNFORMATTED   */
                                        /* SCREEN                 */
                                        /* MAP ONLY               */
    EXEC CICS SEND MAP('DSN8CN2') MAPONLY MAPSET('DSN8CPN') ERASE;
    EXEC CICS RETURN TRANSID('D8PT');
1/*******************************************************************/
 /*             SQL ERROR HANDLING                                  */
 /*******************************************************************/
 P3_DBERROR:                         /* SQL ERROR HANDLING         */
    CALL DSN8MPG (MODULE, '060E', OUTMSG);
    CHAR_SQLCODE = SQLCODE;
    DSN8CN3I.EMSGI = OUTMSG||CHAR_SQLCOD;
    EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ;
 P3_DBERROR2:
    EXEC CICS SEND PAGE ;            /* PERFORM ROLLBACK           */
    EXEC CICS SYNCPOINT ROLLBACK;
    EXEC CICS RETURN;
 P3_CLEAR:                           /* CLEAR SCREEN               */
    EXEC CICS SEND CONTROL FREEKB ;
    EXEC CICS RETURN;
 /*******************************************************************/
 P3_END:                              /*  PROGRAM END              */
    END DSN8CP3;
```

Related reference

“Sample applications in CICS” on page 1378
A set of Db2 sample applications run in the CICS environment.

DSNTEJ5C

THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH COBOL.

```
//********************************************************************* 00010000
//*  NAME = DSNTEJ5C                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 5                                         00050000
//*                     COBOL, CICS                                     00060000
//*                                                                     00070000
//*    Licensed Materials - Property of IBM                             00080000
//*    5635-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2006 IBM Corp.  All Rights Reserved.         00100000
//*                                                                     00110000
//*    STATUS = Version 11                                              00120000
//*                                                                     00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE      00140000
//*             APPLICATIONS AT SITES WITH COBOL.  IT PREPARES THE      00150000
//*             COBOL CICS PROGRAM.                                     00160000
//*                                                                     00170000
//*             RUN THIS JOB ANYTIME AFTER PHASE 2.                     00180000
//*                                                                     00180100
//*  CHANGE ACTIVITY =                                                  00181000
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00182000
//*               Add SET CURRENT SQLID                                 00183001
//*                                                                     00190000
//********************************************************************* 00200000
//JOBLIB   DD  DSN=DSN!!0.SDSNEXIT,DISP=SHR                             00210000
//         DD  DSN=DSN!!0.SDSNLOAD,DISP=SHR                             00220000
//         DD  DSN=CICSTS.SDFHLOAD,DISP=SHR                             00230000
//*                                                                     00240000
//*       STEP  1: CREATE CICS LOGICAL MAP                              00250000
//MAPG    EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',            00260000
//             OUTC='*'                                                 00270000
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MCMG),                        00280000
//             DISP=OLD                                                 00290000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CCG),                            00300000
//             DISP=SHR                                                 00310000
//*                                                                     00320000
//*        STEP  2: CREATE CICS LOGICAL MAP                             00330000
//MAPD    EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',            00340000
//             COND=(4,LT),OUTC='*'                                     00350000
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MCMD),                        00360000
//             DISP=OLD                                                 00370000
```

```
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CCD),                            00380000
//             DISP=SHR                                                 00390000
//*                                                                     00400000
//*        STEP  3: PREPARE CICS COBOL PROGRAMS                         00410000
//DSNH     EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=50                    00420000
//SYSTSPRT DD SYSOUT=*                                                  00430000
//SYSTERM  DD SYSOUT=*                                                  00440000
//SYSPRINT DD SYSOUT=*                                                  00450000
//SYSUDUMP DD SYSOUT=*                                                  00460000
//SYSOUT   DD SYSOUT=*                                                  00470000
//SYSPROC  DD DSN=DSN!!0.SDSNCLST,DISP=SHR                              00480000
//SYSTSIN  DD *                                                         00490000
  %DSNH INPUT('''DSN!!0.SDSNSAMP(DSN8CC0)''') +                         00500000
        PLIB('''DSN!!0.SRCLIB.DATA''') +                                00510000
        P2LIB('''DSN!!0.SDSNSAMP''') +                                  00520000
        TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +                  00530000
        HOST(IBMCOB) RUN(CICS) BIND(NO) +                               00540000
        DELIM(APOST) SQLDELIM(APOSTSQL) +                               00550000
        DBRMLIB('''DSN!!0.DBRMLIB.DATA''')   +                          00560000
        PRELINK(YES) +                                                  00570000
        LLIB('''DSN!!0.RUNLIB.LOAD''')   +                              00580000
        COBICOMP('''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)''') +                 00590000
        COPTION('''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)''') +       00600000
        LOPTION('LIST,XREF,MAP,RENT') +                                 00610000
        STDSQL(NO) +                                                    00620000
        XLIB('''DSN!!0.SDSNLOAD''')  +                                  00630000
        LOAD('''DSN!!0.RUNLIB.LOAD''')                                  00640000
  %DSNH INPUT('''DSN!!0.SDSNSAMP(DSN8CC1)''') +                         00650000
        PLIB('''DSN!!0.SRCLIB.DATA''') +                                00660000
        P2LIB('''DSN!!0.SDSNSAMP''') +                                  00670000
        TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +                  00680000
        HOST(IBMCOB) RUN(CICS) BIND(NO) +                               00690000
        DELIM(APOST) SQLDELIM(APOSTSQL)  +                              00700000
        DBRMLIB('''DSN!!0.DBRMLIB.DATA''')   +                          00710000
        PRELINK(YES) +                                                  00720000
        LLIB('''DSN!!0.RUNLIB.LOAD''')   +                              00730000
        COBICOMP('''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)''') +                 00740000
        COPTION('''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)''') +       00750000
        LOPTION('LIST,XREF,MAP,RENT') +                                 00760000
        STDSQL(NO) +                                                    00770000
        XLIB('''DSN!!0.SDSNLOAD''')  +                                  00780000
        LOAD('''DSN!!0.RUNLIB.LOAD''')                                  00790000
  %DSNH INPUT('''DSN!!0.SDSNSAMP(DSN8CC2)''') +                         00800000
        PLIB('''DSN!!0.SRCLIB.DATA''') +                                00810000
        P2LIB('''DSN!!0.SDSNSAMP''') +                                  00820000
        TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +                  00830000
        HOST(IBMCOB) RUN(CICS) BIND(NO) +                               00840000
        DELIM(APOST) SQLDELIM(APOSTSQL)  +                              00850000
        DBRMLIB('''DSN!!0.DBRMLIB.DATA''')   +                          00860000
        PRELINK(YES) +                                                  00870000
        LLIB('''DSN!!0.RUNLIB.LOAD''')   +                              00880000
        COBICOMP('''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)''') +                 00890000
        COPTION('''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)''') +       00900000
        LOPTION('LIST,XREF,MAP,RENT') +                                 00910000
        STDSQL(NO) +                                                    00920000
        XLIB('''DSN!!0.SDSNLOAD''') +                                   00930000
        LOAD('''DSN!!0.RUNLIB.LOAD''')                                  00940000
//*                                                                     00950000
//*        STEP  4: BIND THE PROGRAM                                    00960000
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    00970000
//DBRMLIB  DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR                          00980000
//SYSUDUMP DD SYSOUT=*                                                  00990000
//SYSTSPRT DD SYSOUT=*                                                  01000000
//SYSPRINT DD SYSOUT=*                                                  01010000
//SYSIN    DD *                                                         01020000
   SET CURRENT SQLID = 'SYSADM';                                        01021001
   GRANT BIND, EXECUTE ON PLAN DSN8CC0                                  01030002
     TO PUBLIC;                                                         01031002
//SYSTSIN  DD *                                                         01040000
DSN SYSTEM(DSN)                                                         01050000
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC0) -                               01060000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 01070000
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC1) -                               01080000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 01090000
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC2) -                               01100000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 01110000
BIND PLAN(DSN8CC0)  PKLIST(DSN8CC!!.*) -                                01120000
     ACTION(REPLACE) RETAIN +                                           01121000
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          01130000
RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  01140000
     LIB('DSN!!0.RUNLIB.LOAD')                                          01150000
END                                                                     01160000
```

```
//*                                                                     01170000
//*        STEP  5: CREATE THE CICS BMS PHYSICAL MAP                    01180000
//MAPGP   EXEC DFHASMVS,COND=(4,LT),OUTC='*'                            01190000
//SYSPUNCH DD  DSN=&&TEMP,                                              01200000
//             DISP=(,PASS),                                            01210000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        01220000
//             DCB=(RECFM=F,BLKSIZE=80)                                 01230000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CCG),                            01240000
//             DISP=SHR                                                 01250000
//*                                                                     01260000
//*        STEP  6: LINKEDIT THE CICS BMS PHYSICAL MAP                  01270000
//MAPGL    EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               01280000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          01290000
//SYSLMOD  DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                           01300000
//SYSPRINT DD SYSOUT=*                                                  01310000
//SYSUDUMP DD SYSOUT=*                                                  01320000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              01330000
//         DD *                                                         01340000
 NAME DSN8CCG(R)                                                        01350000
//*                                                                     01360000
//*        STEP  7: CREATE THE CICS BMS PHYSICAL MAP                    01370000
//MAPDP   EXEC DFHASMVS,COND=(4,LT),OUTC='*'                            01380000
//SYSPUNCH DD  DSN=&&TEMP,                                              01390000
//             DISP=(,PASS),                                            01400000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        01410000
//             DCB=(RECFM=F,BLKSIZE=80)                                 01420000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CCD),                            01430000
//             DISP=SHR                                                 01440000
//*                                                                     01450000
//*        STEP  8: LINKEDIT THE CICS BMS PHYSICAL MAP                  01460000
//MAPDL    EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               01470000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          01480000
//SYSLMOD  DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR                           01490000
//SYSPRINT DD SYSOUT=*                                                  01500000
//SYSUDUMP DD SYSOUT=*                                                  01510000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              01520000
//         DD *                                                         01530000
 NAME DSN8CCD(R)                                                        01540000
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

DSNTEJ5P

THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE APPLICATIONS AT SITES WITH PL/I.

```
//******************************************************************** 00010000
//*  NAME = DSNTEJ5P                                                    00020000
//*                                                                     00030000
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION                          00040000
//*                     PHASE 5                                         00050000
//*                     PL/I, CICS                                      00060000
//*                                                                     00070000
//*    Licensed Materials - Property of IBM                             00080000
//*    5605-DB2                                                         00090000
//*    (C) COPYRIGHT 1982, 2010 IBM Corp.  All Rights Reserved.         00100000
//*                                                                     00110000
//*    STATUS = Version 11                                              00120000
//*                                                                     00130000
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE      00140000
//*             APPLICATIONS AT SITES WITH PL/I.  IT PREPARES THE       00150000
//*             PL/I CICS PROGRAM.                                      00160000
//*                                                                     00170000
//*             RUN THIS JOB ANYTIME AFTER PHASE 2.                     00180000
//*                                                                     00190000
//*  CHANGE ACTIVITY =                                                  00191000
//*    11/07/2012 Add RETAIN to BIND PLAN stmts   dn1651_inst1 / dn1651 00192000
//*               Add SET CURRENT SQLID                                 00192101
//*                                                                     00193000
//******************************************************************** 00200000
//*                                                                     00210000
//JOBLIB   DD  DISP=SHR,DSN=CICSTS.SDFHLOAD                             00220000
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD                             00230000
//*                                                                     00240000
//*        STEP  1: CREATE CICS BMS LOGICAL MAPS                        00250000
//PH05PS01 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',           00260000
//             OUTC='*'                                                 00270000
```

```
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPMG),
//             DISP=OLD
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPG),
//             DISP=SHR
//*
//*        STEP  2: CREATE CICS BMS LOGICAL MAPS
//PH05PS02 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',
//             COND=(4,LT),OUTC='*'
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPMD),
//             DISP=OLD
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPD),
//             DISP=SHR
//*
//*        STEP  3: CREATE CICS BMS LOGICAL MAPS
//PH05PS03 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',
//             COND=(4,LT),OUTC='*'
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPMN),
//             DISP=OLD
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPN),
//             DISP=SHR
//*
//*        STEP  4: CREATE CICS BMS LOGICAL MAPS
//PH05PS04 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',
//             COND=(4,LT),OUTC='*'
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPML),
//             DISP=OLD
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPL),
//             DISP=SHR
//*
//*        STEP  5: CREATE CICS BMS LOGICAL MAPS
//PH05PS05 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',
//             COND=(4,LT),OUTC='*'
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPMU),
//             DISP=OLD
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPU),
//             DISP=SHR
//*
//*        STEP  6: CICS TRANSLATE FOR SQL 0 PART
//PH05PS06 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSPUNCH DD  DSN=&&CICSOUT0,
//             DISP=(NEW,PASS),
//             UNIT=SYSDA,SPACE=(400,(100,100)),
//             DCB=BLKSIZE=400
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CP0),
//             DISP=SHR
//*
//*        STEP  7: PREPARE SQL 0 PART
//PH05PS07 EXEC DSNHPLI,MEM=DSN8CP0,
//         COND=(4,LT),
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
//         PARM.LKED='NCAL'
//PPLI.SYSIN   DD DSN=&&CICSOUT0,DISP=(OLD,DELETE)
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,
//             DISP=SHR
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP0),
//             DISP=SHR
//PC.SYSCIN    DD DSN=&&DSNHOUT0
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,
//             DISP=SHR
//             DD DSN=DSN!!0.SDSNSAMP,
//             DISP=SHR
//PLI.SYSIN    DD DSN=&&DSNHOUT0
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP0),
//             DISP=SHR
//LKED.SYSIN   DD DUMMY
//*
//*        STEP  8: CICS TRANSLATE FOR SQL 1 PART
//PH05PS08 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSPUNCH DD  DSN=&&CICSOUT1,
//             DISP=(NEW,PASS),
//             UNIT=SYSDA,SPACE=(400,(100,100)),
//             DCB=BLKSIZE=400
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CP1),
//             DISP=SHR
//*
```

```
//*        STEP  9: PREPARE SQL 1 PART                                  01100000
//PH05PS09 EXEC DSNHPLI,MEM=DSN8CP1,                                    01110000
//         COND=(4,LT),                                                 01120000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',        01130000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',          01140000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',              01150000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),                01160000
//         PARM.LKED='NCAL'                                             01170000
//PPLI.SYSIN   DD DSN=&&CICSOUT1,DISP=(OLD,DELETE)                      01180000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                   01190000
//             DISP=SHR                                                 01200000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP1),                     01210000
//             DISP=SHR                                                 01220000
//PC.SYSCIN    DD DSN=&&DSNHOUT1                                        01230000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01240000
//             DISP=SHR                                                 01250000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01260000
//             DISP=SHR                                                 01270000
//PLI.SYSIN    DD DSN=&&DSNHOUT1                                        01280000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP1),                      01290000
//             DISP=SHR                                                 01300000
//LKED.SYSIN   DD DUMMY                                                 01310000
//*                                                                     01320000
//*        STEP 10: CICS TRANSLATE FOR SQL 2 PART                       01330000
//PH05PS10 EXEC PGM=DFHEPP1$,COND=(4,LT)                                01340000
//SYSPRINT DD  SYSOUT=*                                                 01350000
//SYSUDUMP DD  SYSOUT=*                                                 01360000
//SYSPUNCH DD  DSN=&&CICSOUT2,                                          01370000
//             DISP=(NEW,PASS),                                         01380000
//             UNIT=SYSDA,SPACE=(400,(100,100)),                        01390000
//             DCB=BLKSIZE=400                                          01400000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CP2),                            01410000
//             DISP=SHR                                                 01420000
//*                                                                     01430000
//*        STEP 11: PREPARE SQL 2 PART                                  01440000
//PH05PS11 EXEC DSNHPLI,MEM=DSN8CP2,                                    01450000
//         COND=(4,LT),                                                 01460000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',        01470000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',          01480000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',              01490000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),                01500000
//         PARM.LKED='NCAL'                                             01510000
//PPLI.SYSIN   DD DSN=&&CICSOUT2,DISP=(OLD,DELETE)                      01520000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                   01530000
//             DISP=SHR                                                 01540000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP2),                     01550000
//             DISP=SHR                                                 01560000
//PC.SYSCIN    DD DSN=&&DSNHOUT2                                        01570000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               01580000
//             DISP=SHR                                                 01590000
//             DD DSN=DSN!!0.SDSNSAMP,                                  01600000
//             DISP=SHR                                                 01610000
//PLI.SYSIN    DD DSN=&&DSNHOUT2                                        01620000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP2),                      01630000
//             DISP=SHR                                                 01640000
//LKED.SYSIN   DD DUMMY                                                 01650000
//*                                                                     01660000
//*        STEP 12: CICS TRANSLATE FOR TELEPHONE APPLICATION            01670000
//PH05PS12 EXEC PGM=DFHEPP1$,COND=(4,LT)                                01680000
//SYSPRINT DD SYSOUT=*                                                  01690000
//SYSUDUMP DD SYSOUT=*                                                  01700000
//SYSPUNCH DD  DSN=&&CICSOUT3,                                          01710000
//             DISP=(NEW,PASS),                                         01720000
//             UNIT=SYSDA,SPACE=(400,(100,100)),                        01730000
//             DCB=BLKSIZE=400                                          01740000
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8CP3),                             01750000
//             DISP=SHR                                                 01760000
//*                                                                     01770000
//*        STEP 13: PREPARE TELEPHONE APPLICATION                       01780000
//PH05PS13 EXEC DSNHPLI,MEM=DSN8CP3,                                    01790000
//         COND=(4,LT),                                                 01800000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',        01810000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',          01820000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',              01830000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),                01840000
//         PARM.LKED='NCAL'                                             01850000
//PPLI.SYSIN   DD DSN=&&CICSOUT3,DISP=(OLD,DELETE)                      01860000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                   01870000
//             DISP=SHR                                                 01880000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP3),                     01890000
//             DISP=SHR                                                 01900000
//PC.SYSCIN    DD DSN=&&DSNHOUT3                                        01910000
```

```
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                             01920000
//              DISP=SHR                                              01930000
//              DD DSN=DSN!!0.SDSNSAMP,                               01940000
//              DISP=SHR                                              01950000
//PLI.SYSIN    DD DSN=&&DSNHOUT3                                      01960000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP3),                    01970000
//              DISP=SHR                                              01980000
//LKED.SYSIN   DD DUMMY                                               01990000
//*                                                                   02000000
//*        STEP 14: CREATE CICS BMS LOGICAL MAPS                      02010000
//PH05PS14 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',         02020000
//              COND=(4,LT),OUTC='*'                                  02030000
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPMF),                      02040000
//              DISP=OLD                                              02050000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPF),                          02060000
//              DISP=SHR                                              02070000
//*                                                                   02080000
//*        STEP 15: CREATE CICS BMS LOGICAL MAPS                      02090000
//PH05PS15 EXEC DFHASMVS,PARM='DECK,NOOBJECT,SYSPARM(DSECT)',         02100000
//              COND=(4,LT),OUTC='*'                                  02110000
//SYSPUNCH DD  DSN=DSN!!0.SRCLIB.DATA(DSN8MPME),                      02120000
//              DISP=OLD                                              02130000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPE),                          02140000
//              DISP=SHR                                              02150000
//*                                                                   02160000
//*        STEP 16: CICS TRANSLATE FOR SQL 0 PART                     02170000
//PH05PS16 EXEC PGM=DFHEPP1$,COND=(4,LT)                              02180000
//SYSPRINT DD  SYSOUT=*                                               02190000
//SYSUDUMP DD  SYSOUT=*                                               02200000
//SYSPUNCH DD  DSN=&&CICSOUT6,                                        02210000
//              DISP=(NEW,PASS),                                      02220000
//              UNIT=SYSDA,SPACE=(400,(100,100)),                     02230000
//              DCB=BLKSIZE=400                                       02240000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CP6),                          02250000
//              DISP=SHR                                              02260000
//*                                                                   02270000
//*        STEP 17: PREPARE SQL 0 PART                                02280000
//PH05PS17 EXEC DSNHPLI,MEM=DSN8CP6,                                  02290000
//         COND=(4,LT),                                               02300000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      02310000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',        02320000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            02330000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),              02340000
//         PARM.LKED='NCAL'                                           02350000
//PPLI.SYSIN   DD DSN=&&CICSOUT6,DISP=(OLD,DELETE)                    02360000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                 02370000
//              DISP=SHR                                              02380000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP6),                   02390000
//              DISP=SHR                                              02400000
//PC.SYSCIN    DD DSN=&&DSNHOUT6                                      02410000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                             02420000
//              DISP=SHR                                              02430000
//              DD DSN=DSN!!0.SDSNSAMP,                               02440000
//              DISP=SHR                                              02450000
//PLI.SYSIN    DD DSN=&&DSNHOUT6                                      02460000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP6),                    02470000
//              DISP=SHR                                              02480000
//LKED.SYSIN   DD DUMMY                                               02490000
//*                                                                   02500000
//*        STEP 18: CICS TRANSLATE FOR SQL 1 PART                     02510000
//PH05PS18 EXEC PGM=DFHEPP1$,COND=(4,LT)                              02520000
//SYSPRINT DD  SYSOUT=*                                               02530000
//SYSUDUMP DD  SYSOUT=*                                               02540000
//SYSPUNCH DD  DSN=&&CICSOUT7,                                        02550000
//              DISP=(NEW,PASS),                                      02560000
//              UNIT=SYSDA,SPACE=(400,(100,100)),                     02570000
//              DCB=BLKSIZE=400                                       02580000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CP7),                          02590000
//              DISP=SHR                                              02600000
//*                                                                   02610000
//*        STEP 19: PREPARE SQL 1 PART                                02620000
//PH05PS19 EXEC DSNHPLI,MEM=DSN8CP7,                                  02630000
//         COND=(4,LT),                                               02640000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',      02650000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',        02660000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',            02670000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),              02680000
//         PARM.LKED='NCAL'                                           02690000
//PPLI.SYSIN   DD DSN=&&CICSOUT7,DISP=(OLD,DELETE)                    02700000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                 02710000
//              DISP=SHR                                              02720000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP7),                   02730000
```

```
//              DISP=SHR                                                02740000
//PC.SYSCIN    DD DSN=&&DSNHOUT7                                        02750000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               02760000
//              DISP=SHR                                                02770000
//              DD DSN=DSN!!0.SDSNSAMP,                                 02780000
//              DISP=SHR                                                02790000
//PLI.SYSIN    DD DSN=&&DSNHOUT7                                        02800000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP7),                      02810000
//              DISP=SHR                                                02820000
//LKED.SYSIN   DD DUMMY                                                 02830000
//*                                                                     02840000
//*        STEP 20: CICS TRANSLATE FOR SQL 2 PART                       02850000
//PH05PS20 EXEC PGM=DFHEPP1$,COND=(4,LT)                                02860000
//SYSPRINT DD SYSOUT=*                                                  02870000
//SYSUDUMP DD SYSOUT=*                                                  02880000
//SYSPUNCH DD  DSN=&&CICSOUT8,                                          02890000
//              DISP=(NEW,PASS),                                        02900000
//              UNIT=SYSDA,SPACE=(400,(100,100)),                       02910000
//              DCB=BLKSIZE=400                                         02920000
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8CP8),                             02930000
//              DISP=SHR                                                02940000
//*                                                                     02950000
//*        STEP 21: PREPARE SQL 2 PART                                  02960000
//PH05PS21 EXEC DSNHPLI,MEM=DSN8CP8,                                    02970000
//         COND=(4,LT),                                                 02980000
//         PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',        02990000
//         PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',          03000000
//         PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',              03010000
//         'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),                03020000
//         PARM.LKED='NCAL'                                             03030000
//PPLI.SYSIN   DD DSN=&&CICSOUT8,DISP=(OLD,DELETE)                      03040000
//PPLI.SYSLIB  DD DSN=CICSTS.SDFHPL1,                                   03050000
//              DISP=SHR                                                03060000
//PC.DBRMLIB   DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP8),                     03070000
//              DISP=SHR                                                03080000
//PC.SYSCIN    DD DSN=&&DSNHOUT8                                        03090000
//PC.SYSLIB    DD DSN=DSN!!0.SRCLIB.DATA,                               03100000
//              DISP=SHR                                                03110000
//              DD DSN=DSN!!0.SDSNSAMP,                                 03120000
//              DISP=SHR                                                03130000
//PLI.SYSIN    DD DSN=&&DSNHOUT8                                        03140000
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP8),                      03150000
//              DISP=SHR                                                03160000
//LKED.SYSIN   DD DUMMY                                                 03170000
//*                                                                     03180000
//*        STEP 22: LINKEDIT PROGRAMS TOGETHER                          03190000
//PH05PS22 EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(4,LT)               03200000
//SYSLIB   DD  DSN=CEE.V!R!M!.SCEELKED,                                 03210000
//              DISP=SHR                                                03220000
//         DD  DSN=DSN!!0.SDSNLOAD,                                     03230000
//              DISP=SHR                                                03240000
//         DD  DSN=CICSTS.SDFHPL1,                                      03250000
//              DISP=SHR                                                03260000
//         DD  DSN=CICSTS.SDFHLOAD,                                     03270000
//              DISP=SHR                                                03280000
//SYSLMOD  DD  DSN=DSN!!0.RUNLIB.LOAD,                                  03290000
//              DISP=SHR                                                03300000
//SYSPRINT DD  SYSOUT=*                                                 03310000
//SYSUDUMP DD  SYSOUT=*                                                 03320000
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))                          03330000
//SYSLIN   DD  *                                                        03340000
    INCLUDE SYSLIB(CEESTART)                                            03350000
    INCLUDE SYSLIB(CEESG010)                                            03360000
    INCLUDE SYSLIB(DFHELII)                                             03370000
    INCLUDE SYSLIB(DSNCLI)                                              03380000
    REPLACE PLISTART                                                    03390000
    CHANGE  PLIMAIN(CEEMAIN)                                            03400000
    INCLUDE SYSLMOD(DSN8CP0)                                            03410000
    INCLUDE SYSLMOD(DSN8MPG)                                            03420000
    ORDER   CEESTART                                                    03430000
    ENTRY   CEESTART                                                    03440000
     NAME   DSN8CP0(R)                                                  03450000
    INCLUDE SYSLIB(CEESTART)                                            03460000
    INCLUDE SYSLIB(CEESG010)                                            03470000
    INCLUDE SYSLIB(DFHELII)                                             03480000
    INCLUDE SYSLIB(DSNCLI)                                              03490000
    REPLACE PLISTART                                                    03500000
    CHANGE  PLIMAIN(CEEMAIN)                                            03510000
    INCLUDE SYSLMOD(DSN8CP1)                                            03520000
    INCLUDE SYSLMOD(DSN8MPG)                                            03530000
    ORDER   CEESTART                                                    03540000
    ENTRY   CEESTART                                                    03550000
```

```
     NAME   DSN8CP1(R)                                                  03560000
    INCLUDE SYSLIB(CEESTART)                                            03570000
    INCLUDE SYSLIB(CEESG010)                                            03580000
    INCLUDE SYSLIB(DFHELII)                                             03590000
    INCLUDE SYSLIB(DSNCLI)                                              03600000
    REPLACE PLISTART                                                    03610000
    CHANGE  PLIMAIN(CEEMAIN)                                            03620000
    INCLUDE SYSLMOD(DSN8CP2)                                            03630000
    INCLUDE SYSLMOD(DSN8MPG)                                            03640000
    ORDER   CEESTART                                                    03650000
    ENTRY   CEESTART                                                    03660000
     NAME   DSN8CP2(R)                                                  03670000
    INCLUDE SYSLIB(CEESTART)                                            03680000
    INCLUDE SYSLIB(CEESG010)                                            03690000
    INCLUDE SYSLIB(DFHELII)                                             03700000
    INCLUDE SYSLIB(DSNCLI)                                              03710000
    REPLACE PLISTART                                                    03720000
    CHANGE  PLIMAIN(CEEMAIN)                                            03730000
    INCLUDE SYSLMOD(DSN8CP3)                                            03740000
    INCLUDE SYSLMOD(DSN8MPG)                                            03750000
    ORDER   CEESTART                                                    03760000
    ENTRY   CEESTART                                                    03770000
     NAME   DSN8CP3(R)                                                  03780000
    INCLUDE SYSLIB(CEESTART)                                            03790000
    INCLUDE SYSLIB(CEESG010)                                            03800000
    INCLUDE SYSLIB(DFHELII)                                             03810000
    INCLUDE SYSLIB(DSNCLI)                                              03820000
    REPLACE PLISTART                                                    03830000
    CHANGE  PLIMAIN(CEEMAIN)                                            03840000
    INCLUDE SYSLMOD(DSN8CP6)                                            03850000
    INCLUDE SYSLMOD(DSN8MPG)                                            03860000
    ORDER   CEESTART                                                    03870000
    ENTRY   CEESTART                                                    03880000
     NAME   DSN8CP6(R)                                                  03890000
    INCLUDE SYSLIB(CEESTART)                                            03900000
    INCLUDE SYSLIB(CEESG010)                                            03910000
    INCLUDE SYSLIB(DFHELII)                                             03920000
    INCLUDE SYSLIB(DSNCLI)                                              03930000
    REPLACE PLISTART                                                    03940000
    CHANGE  PLIMAIN(CEEMAIN)                                            03950000
    INCLUDE SYSLMOD(DSN8CP7)                                            03960000
    INCLUDE SYSLMOD(DSN8MPG)                                            03970000
    ORDER   CEESTART                                                    03980000
    ENTRY   CEESTART                                                    03990000
     NAME   DSN8CP7(R)                                                  04000000
    INCLUDE SYSLIB(CEESTART)                                            04010000
    INCLUDE SYSLIB(CEESG010)                                            04020000
    INCLUDE SYSLIB(DFHELII)                                             04030000
    INCLUDE SYSLIB(DSNCLI)                                              04040000
    REPLACE PLISTART                                                    04050000
    CHANGE  PLIMAIN(CEEMAIN)                                            04060000
    INCLUDE SYSLMOD(DSN8CP8)                                            04070000
    INCLUDE SYSLMOD(DSN8MPG)                                            04080000
    ORDER   CEESTART                                                    04090000
    ENTRY   CEESTART                                                    04100000
     NAME   DSN8CP8(R)                                                  04110000
//*                                                                     04120000
//*        STEP 23: BIND PROGRAMS                                       04130000
//PH05PS23 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)                    04140000
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA                         04150000
//SYSUDUMP DD SYSOUT=*                                                  04160000
//SYSTSPRT DD SYSOUT=*                                                  04170000
//SYSPRINT DD SYSOUT=*                                                  04180000
//SYSIN    DD *                                                         04190000
   SET CURRENT SQLID = 'SYSADM';                                        04191001
   GRANT BIND, EXECUTE ON PLAN DSN8CP0, DSN8CQ0, DSN8CH0                04200002
     TO PUBLIC;                                                         04201002
//SYSTSIN  DD *                                                         04210000
DSN SYSTEM(DSN)                                                         04220000
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP0) +                                04281000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04282000
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP1) +                                04283000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04284000
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP2) +                                04285000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04286000
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP3) +                                04287000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04288000
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP6) +                                04289000
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04289100
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP7) +                                04289200
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04289300
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP8) +                                04289400
```

```
     ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                 04289500
BIND PLAN(DSN8CP0) +                                                    04289603
     PKLIST(DSN8CP!!.DSN8CP0, +                                         04289703
            DSN8CP!!.DSN8CP1, +                                         04289803
            DSN8CP!!.DSN8CP2) +                                         04289903
     ACTION(REPLACE) RETAIN +                                           04290000
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          04290100
BIND PLAN(DSN8CQ0) +                                                    04290203
     PKLIST(DSN8CP!!.DSN8CP6, +                                         04290303
            DSN8CP!!.DSN8CP7, +                                         04290403
            DSN8CP!!.DSN8CP8) +                                         04290503
     ACTION(REPLACE) RETAIN +                                           04290600
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          04290700
BIND PLAN(DSN8CH0) +                                                    04290803
     PKLIST(DSN8CP!!.DSN8CP3) +                                         04290903
     ACTION(REPLACE) RETAIN +                                           04291000
     ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)                          04291100
RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -                                  04292000
     LIB('DSN!!0.RUNLIB.LOAD')                                          04300000
END                                                                     04310000
//*                                                                     04320000
//*        STEP 24: CREATE CICS BMS PHYSICAL MAPS                       04330000
//PH05PS24 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           04340000
//SYSPUNCH DD  DSN=&&TEMP,                                              04350000
//             DISP=(NEW,PASS),                                         04360000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        04370000
//             DCB=(RECFM=F,BLKSIZE=80)                                 04380000
//SYSIN    DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSN8CPG)                     04390000
//*                                                                     04400000
//*        STEP 25: LINKEDIT CICS BMS PHYSICAL MAPS                     04410000
//PH05PS25 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               04420000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          04430000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           04440000
//SYSPRINT DD SYSOUT=*                                                  04450000
//SYSUDUMP DD SYSOUT=*                                                  04460000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              04470000
//         DD *                                                         04480000
 NAME DSN8CPG(R)                                                        04490000
//*                                                                     04500000
//*        STEP 26: CREATE CICS BMS PHYSICAL MAPS                       04510000
//PH05PS26 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           04520000
//SYSPUNCH DD  DSN=&&TEMP,                                              04530000
//             DISP=(NEW,PASS),                                         04540000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        04550000
//             DCB=(RECFM=F,BLKSIZE=80)                                 04560000
//SYSIN    DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSN8CPD)                     04570000
//*                                                                     04580000
//*        STEP 27: LINKEDIT CICS BMS PHYSICAL MAPS                     04590000
//PH05PS27 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               04600000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          04610000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           04620000
//SYSPRINT DD SYSOUT=*                                                  04630000
//SYSUDUMP DD SYSOUT=*                                                  04640000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              04650000
//         DD *                                                         04660000
 NAME DSN8CPD(R)                                                        04670000
//*                                                                     04680000
//*        STEP 28: CREATE CICS BMS PHYSICAL MAPS                       04690000
//PH05PS28 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           04700000
//SYSPUNCH DD  DSN=&&TEMP,                                              04710000
//             DISP=(NEW,PASS),                                         04720000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        04730000
//             DCB=(RECFM=F,BLKSIZE=80)                                 04740000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPN),                            04750000
//             DISP=SHR                                                 04760000
//*                                                                     04770000
//*        STEP 29: LINKEDIT CICS BMS PHYSICAL MAPS                     04780000
//PH05PS29 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               04790000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          04800000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           04810000
//SYSPRINT DD SYSOUT=*                                                  04820000
//SYSUDUMP DD SYSOUT=*                                                  04830000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              04840000
//         DD *                                                         04850000
 NAME DSN8CPN(R)                                                        04860000
//*                                                                     04870000
//*        STEP 30: CREATE CICS BMS PHYSICAL MAPS                       04880000
//PH05PS30 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           04890000
//SYSPUNCH DD  DSN=&&TEMP,                                              04900000
//             DISP=(NEW,PASS),                                         04910000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        04920000
//             DCB=(RECFM=F,BLKSIZE=80)                                 04930000
```

```
//SYSIN    DD DSN=DSN!!0.SDSNSAMP(DSN8CPL),                             04940000
//             DISP=SHR                                                 04950000
//*                                                                     04960000
//*        STEP 31: LINKEDIT CICS BMS PHYSICAL MAPS                     04970000
//PH05PS31 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               04980000
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(100,10))                         04990000
//SYSLMOD  DD  DSN=DSN!!0.RUNLIB.LOAD,                                  05000000
//             DISP=SHR                                                 05010000
//SYSPRINT DD  SYSOUT=*                                                 05020000
//SYSUDUMP DD  SYSOUT=*                                                 05030000
//SYSLIN   DD  DSN=&&TEMP,DISP=(OLD,DELETE)                             05040000
//         DD  *                                                        05050000
 NAME DSN8CPL(R)                                                        05060000
//*                                                                     05070000
//*        STEP 32: CREATE CICS BMS PHYSICAL MAPS                       05080000
//PH05PS32 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           05090000
//SYSPUNCH DD  DSN=&&TEMP,                                              05100000
//             DISP=(NEW,PASS),                                         05110000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        05120000
//             DCB=(RECFM=F,BLKSIZE=80)                                 05130000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPU),                            05140000
//             DISP=SHR                                                 05150000
//*                                                                     05160000
//*        STEP 33: LINKEDIT CICS BMS PHYSICAL MAPS                     05170000
//PH05PS33 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               05180000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          05190000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           05200000
//SYSPRINT DD SYSOUT=*                                                  05210000
//SYSUDUMP DD SYSOUT=*                                                  05220000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              05230000
//         DD *                                                         05240000
 NAME DSN8CPU(R)                                                        05250000
//*                                                                     05260000
//*        STEP 34: CREATE CICS BMS PHYSICAL MAPS                       05270000
//PH05PS34 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           05280000
//SYSPUNCH  DD DSN=&&TEMP,                                              05290000
//             DISP=(NEW,PASS),                                         05300000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        05310000
//             DCB=(RECFM=F,BLKSIZE=80)                                 05320000
//SYSIN     DD DSN=DSN!!0.SDSNSAMP(DSN8CPF),                            05330000
//             DISP=SHR                                                 05340000
//*                                                                     05350000
//*        STEP 35: LINKEDIT CICS BMS PHYSICAL MAPS                     05360000
//PH05PS35 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               05370000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          05380000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           05390000
//SYSPRINT DD SYSOUT=*                                                  05400000
//SYSUDUMP DD SYSOUT=*                                                  05410000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              05420000
//         DD *                                                         05430000
 NAME DSN8CPF(R)                                                        05440000
//*                                                                     05450000
//*        STEP 36: CREATE CICS BMS PHYSICAL MAPS                       05460000
//PH05PS36 EXEC DFHASMVS,COND=(4,LT),OUTC='*'                           05470000
//SYSPUNCH DD  DSN=&&TEMP,                                              05480000
//             DISP=(NEW,PASS),                                         05490000
//             UNIT=SYSDA,SPACE=(1024,(100,10)),                        05500000
//             DCB=(RECFM=F,BLKSIZE=80)                                 05510000
//SYSIN    DD  DSN=DSN!!0.SDSNSAMP(DSN8CPE),                            05520000
//             DISP=SHR                                                 05530000
//*                                                                     05540000
//*        STEP 37: LINKEDIT CICS BMS PHYSICAL MAPS                     05550000
//PH05PS37 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)               05560000
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10))                          05570000
//SYSLMOD  DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD                           05580000
//SYSPRINT DD SYSOUT=*                                                  05590000
//SYSUDUMP DD SYSOUT=*                                                  05600000
//SYSLIN   DD DSN=&&TEMP,DISP=(OLD,DELETE)                              05610000
//         DD *                                                         05620000
 NAME DSN8CPE(R)                                                        05630000
```

Related reference

"Sample applications in CICS" on page 1378
A set of Db2 sample applications run in the CICS environment.

Information resources for Db2 11 for z/OS and related products

Information about Db2 11 for z/OS and products that you might use in conjunction with Db2 11 is available online in IBM Documentation.

You can find the complete set of product documentation for Db2 11 for z/OS in IBM Documentation.

You can also download other PDF format manuals for Db2 11 for z/OS from IBM Documentation in PDF format manuals for Db2 11 for z/OS (Db2 for z/OS in IBM Documentation).

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you to write programs that contain SQL statements. This information primarily documents General-use Programming Interface and Associated Guidance Information provided by Db2 11 for z/OS. However, this information also documents Product-sensitive Programming Interface and Associated Guidance Information provided by Db2 11 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of Db2 11 for z/OS.

Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

PSPI Product-sensitive Programming Interface and Associated Guidance Information... PSPI

Trademarks

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of Db2 for z/OS terms.

Index

Special Characters

Numerics

A

B

C

D

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Product Number: 5615-DB2
5697-P43

SC19-4051-08