DB2 Universal Database for z/OS

Version 8

Release Planning Guide

Information Management software

SC18-7425-05

DB2 Universal Database for z/OS

Version 8

Release Planning Guide

Information Management software

SC18-7425-05

Note
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 149.

Sixth Edition, Softcopy Only (February 2008)

This edition applies to Version 8 of IBM DB2 Universal Database for z/OS (DB2 UDB for z/OS), product number 5625-DB2, 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.

This softcopy version is based on the printed edition of the book and includes the changes indicated in the printed version by vertical bars. Additional changes made to this softcopy version of the book since the hardcopy book was published are indicated by the hash (#) symbol in the left-hand margin. Editorial changes that have no technical significance are not noted.

This and other books in the DB2 UDB for z/OS library are periodically updated with technical changes. These updates are made available to licensees of the product on CD-ROM and on the Web (currently at www.ibm.com/software/data/db2/zos/library.html). Check these resources to ensure that you are using the most current information.

Contents

About this book

DB2 Release Planning Guide is intended to help you plan for Version 8 of the licensed program DB2 Universal Database™ for z/OS®.

Unless it is stated otherwise, this information assumes that DB2® is running in new-function mode (as opposed to compatibility mode or enabling-new-function mode).

Important

In this version of DB2 UDB for z/OS, the DB2 Utilities Suite 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. See Part 1 of *DB2 Utility Guide and Reference* for packaging details.

The DB2 Utilities Suite is designed to work with the DFSORT program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES
- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

Who should read this book

This book is intended for all users of DB2, including application programmers, database administrators, and system programmers. It assumes that the user is familiar with Version 7 of DB2 UDB for z/OS and OS/390®. For more information about how to obtain DB2 information, see Appendix H, “How to obtain DB2 information,” on page 147.

Terminology and citations

In this information, DB2 Universal Database for z/OS is referred to as "DB2 UDB for z/OS." In cases where the context makes the meaning clear, DB2 UDB for z/OS is referred to as "DB2." When this information refers to titles of books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM® DB2 Universal Database for z/OS SQL Reference*.)

When referring to a DB2 product other than DB2 UDB 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.

OMEGAMON

Refers to any of the following products:

- IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

• IBM Tivoli OMEGAMON XE for 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 or CICS Transaction Server for OS/390.

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

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products, including DB2 UDB for z/OS, enable users to:

- Use assistive technologies such as screen reader and screen magnifier software
- Operate specific or equivalent features by using only a keyboard
- Customize display attributes such as color, contrast, and font size

Assistive technology products, such as screen readers, function with the DB2 UDB for z/OS user interfaces. Consult the documentation for the assistive technology products for specific information when you use assistive technology to access these interfaces.

Online documentation for Version 8 of DB2 UDB for z/OS is available in the Information management software for z/OS solutions information center, which is an accessible format when used with assistive technologies such as screen reader or screen magnifier software. The Information management software for z/OS solutions information center is available at the following Web site: http://publib.boulder.ibm.com/infocenter/dzichelp

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 UDB for z/OS documentation. You can use the following methods to provide comments:

- Send your comments by e-mail to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).
- You can send comments from the Web. Visit the library Web site at:

  www.ibm.com/software/db2zos/library.html

  This Web site has a an online reader comment form that you can use to send comments.

- You can also send comments by using the feedback link at the footer of each page in the Information Management Software for z/OS Solutions Information Center at http://publib.boulder.ibm.com/infocenter/db2zhelp.

Chapter 1. Availability, scalability, and performance enhancements

Version 8 of DB2 UDB for z/OS provides functional enhancements to availability, scalability, and performance. The following topics provide additional information:

Changes to limits for better availability, scalability, and performance

Version 8 of DB2 UDB for z/OS provides increased limits for better availability, scalability, and performance as highlighted in Table 1.

Table 1. Changes to limits in DB2 UDB for z/OS, Version 8. This table lists the entities that have changed in Version 8 and their associated limits, both previous and new.

Entity	Previous limit	New limit
Virtual storage	31-bit	64-bit
Length of an index key	255 bytes	2000 bytes
Number of partitions in a partitioned table space or a partitioned index space	254	4096
Number of active logs	31	93
Number of archive logs	1000 per copy of the log 2000 for dual logging	10 000 per copy of the log 20 000 for dual logging
Maximum size of a partitioned table with page size of 8 KB[1]	16 TB	32 TB
Maximum size of a partitioned table with page size of 16 KB	16 TB	64 TB
Maximum size of a partitioned table with page size of 32 KB	16 TB	128 TB

Note:

1. The maximum size of a partitioned table with page size of 4 KB has not changed; the maximum size is 16 TB.

Schema evolution

DB2 UDB for z/OS now provides the ability to change the definitions of tables and indexes without dropping and re-creating the object. This capability significantly enhances both the availability of your database and the performance of data access.

Using the ALTER TABLE statement: With the ALTER TABLE statement, you can change the definition of a table or the partitioning of a table space in the following ways:

- Add a new partition to a table space.
- Rotate the partition with the lowest limit value for reuse as the partition with the highest limit value.
- Change the boundary between partitions or extend the boundary of the last partition.
- Change the data type of a column with the exception of a distinct type column, a LOB column, a column referenced in a field procedure, or a column in a materialized query table.
- Change all of the attributes of an identity column except the data type.
- Add or drop a parent key or a foreign key.
- Add or drop a table check constraint.
- Add a new column to a table.
- Add or drop a clustering index to a table.

When you do not use the ALTER statement to change a table definition, you must:

1. Use the DROP statement to remove the table.
2. Use the COMMIT statement to commit the removal of the table.
3. Use the CREATE statement to re-create the table.

The DROP statement has a cascading effect; views that are dependent on the dropped table are also dropped. All authorities for the dropped objects disappear, and DB2 marks plans or packages that reference dropped objects as invalid.

You can make certain changes only by dropping the table and then re-creating it with the new definition. For example, to change an identity column to a column with a different data type, you must drop and re-create the table with the new definition instead of using the ALTER statement.

Using the ALTER INDEX statement: With the ALTER INDEX statement, you can change the definition of an index in the following ways:

- Add a new column to an index.
- Change how varying-length columns are stored in the index (as padded or not padded), which might increase the possibility of DB2 choosing index-only access.
- Change the clustering attribute of an index to change the location of rows in the table on which the index is defined.

For more information about index enhancements, see “Index enhancements” on page 12.

Ability to use table-controlled partitioning

Before Version 8 of DB2, when you defined a partitioning index on a table in a partitioned table space, you specified the partitioning key and the limit key values in the PART VALUES clause of the CREATE INDEX statement. This type of partitioning is referred to as *index-controlled partitioning*.

With Version 8, you can specify the partitioning key and the limit key values for a table in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement. This type of partitioning is referred to as *table-controlled partitioning*.

If you drop an index that is defined with the PARTITION ENDING AT (previously PART VALUES) clause, DB2 automatically converts the associated index-controlled partitioned table space to a table-controlled partitioned table space. However, if you use the ALTER statement to do any of the following tasks, DB2 automatically converts the table space to table-controlled partitioning, but it does not drop any indexes:

- Add a partition
- Change a partition boundary
- Rotate a partition from first to last
- Create a data-partitioned secondary index
- Specify CLUSTER NO for the partitioning or clustering index

For information about using the ALTER statement to manage partitions, see "Ability to add partitions" on page 6 and "Ability to rotate partitions" on page 6.

Creating new tables with table-controlled partitioning

You can specify the partitioning key and the limit key values for a table in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement. If you use this type of partitioning, you cannot use the PARTITION ENDING AT clause of the CREATE INDEX statement when you create indexes on the table.

Example: Assume that you need to create a large transaction table that includes the date of the transaction in a column named POSTED. You want to keep the transactions for each month in a separate partition. To create the table, issue the following statement:

```
CREATE TABLE TRANS
   (ACCTID ...,
    STATE ...,
    POSTED ...,
    ... , ...)
   PARTITION BY (POSTED)
   (PARTITION 1 ENDING AT ('01/31/2003'),
    PARTITION 2 ENDING AT ('02/28/2003'),
    ...
    PARTITION 13 ENDING AT ('01/31/2004'));
```

Separation of partitioning and clustering

In previous releases of DB2, a partitioned table space could have only one partitioned index, and the partitioned index was the partitioning index as well as the clustering index. As explained in "Schema evolution" on page 2, Version 8 of DB2 introduces *table-controlled partitioning*, in which a table, instead of an index on the table, determines the partitioning scheme. With table-controlled partitioning, the partitioning index is optional. You can assign the clustering attribute to a secondary index, or you can let DB2 assign the clustering attribute. To let DB2

assign the clustering attribute, do not assign the clustering attribute to any index. In addition, you can remove the clustering attribute from one index and assign it to another index. Use the CLUSTER and NOT CLUSTER parameters of CREATE INDEX and ALTER INDEX to accomplish these tasks.

When a data-partitioned secondary index is the clustering index, after a REORG, the data rows are ordered within each partition to match the ordering of the data-partitioned secondary index keys.

Example: Running REORG to reorder data rows: Suppose that you alter index SALES_IX, which is shown in Figure 2 on page 13, like this:

```
ALTER INDEX SALES_IX CLUSTER;
```

After you run the REORG utility, the data looks as shown in Figure 1.

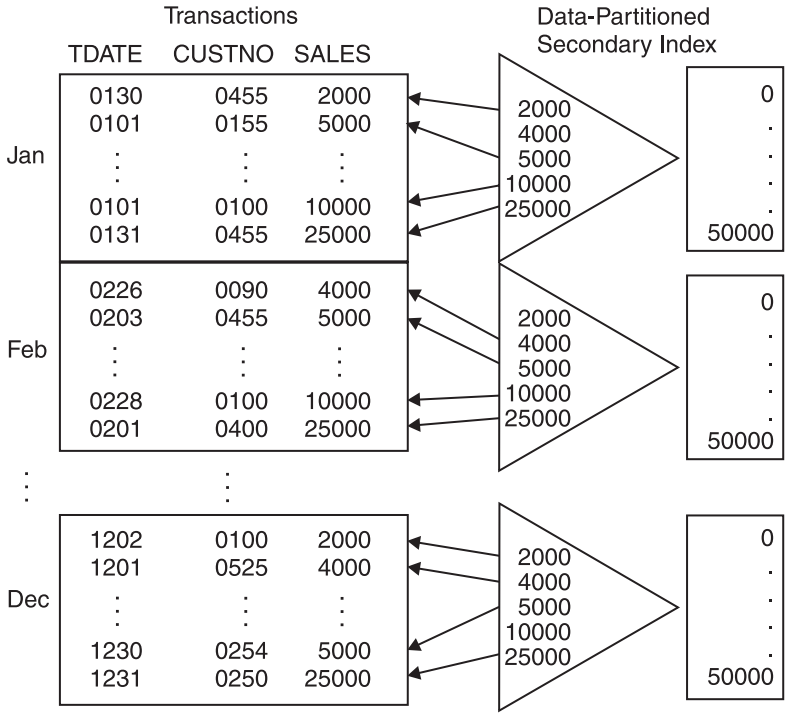

Figure 1. Example of a data-partitioned secondary index as a clustering index

If no explicit clustering index is specified for a table, the first index that is created on a table is the implicit clustering index. If an index is altered from CLUSTER to NOT CLUSTER, that index is still used as the implicit clustering index until a new explicit clustering index is specified. When the clustering index is changed, INSERT statements place new rows in the new clustering order. However, existing data rows are not affected until a REORG utility job runs and places those rows in clustering order.

Clustering within partitions

You can specify any index as the clustering index, regardless of whether it is a partitioning index.

Example: suppose that the TRANS table is partitioned by the DATE column, as described in "Creating new tables with table-controlled partitioning" on page 3. You want to cluster the rows of each partition by the values in the ACCTID column. Issue the statement:

```
CREATE INDEX IX3
  ON TRANS (ACCTID)
  CLUSTER;
```

The rows of the TRANS table are clustered by account number. Each partition contains the account numbers for the transactions during that month, and those account numbers are clustered within each partition.

For more information about clustering that is separated from partitioning, see "Separation of partitioning and clustering" on page 3.

Improving index usage for partitioned table spaces

By changing the way indexes are defined for a table, you can:

- Change an existing index-controlled partitioned table space to a table-controlled partitioned table space.
- Implement a partitioned clustering index so that the index clusters the data within each partition. The new index that is created in the following example is a *data-partitioned secondary index*. For more information about data-partitioned secondary indexes, see "Data-partitioned secondary indexes" on page 12.

Example: Assume that you have a large transaction table named TRANS that contains one row for each transaction. The table includes the following columns:

- ACCTID, which is the customer account ID
- POSTED, which holds the date of the transaction

The table space that contains TRANS is divided into 13 partitions, each of which contains one month of data. Two existing indexes are defined as follows:

- A partitioning index is defined on the transaction date by the following CREATE INDEX statement with a PARTITION ENDING AT clause:

  ```
  CREATE INDEX IX1 ON TRANS(POSTED)
     CLUSTER
     (PARTITION 1 ENDING AT ('01/31/2003'),
      PARTITION 2 ENDING AT ('02/28/2003'),
      ...
      PARTITION 13 ENDING AT ('01/31/2004'));
  ```

  The partitioning index is the clustering index by definition, and the data rows in the table are in order by the transaction date. The partitioning index controls the partitioning of the data in the table space.
- A nonpartitioning index is defined on the customer account ID:

  ```
  CREATE INDEX IX2 ON TRANS(ACCTID);
  ```

  DB2 usually accesses the transaction table through the customer account ID by using the nonpartitioning index IX2.

The partitioning index IX1 is not used for data access and is wasting space. In addition, you have a critical requirement for availability on the table, and you want to be able to run an online REORG job at the partition level with minimal disruption to data availability.

To save space and to facilitate reorganization of the table space, you can drop the partitioning index IX1, and you can replace the access index IX2 with a partitioned clustering index that matches the 13 data partitions in the table. Issue the following statements:

```
DROP INDEX IX1;
CREATE INDEX IX3
  ON TRANS(ACCTID)
  PARTITIONED CLUSTER;
COMMIT;

DROP INDEX IX2;
COMMIT;
```

When you drop the partitioning index IX1, DB2 converts the table space from index-controlled partitioning to table-controlled partitioning. DB2 uses the PARTITION limit key values of the index-controlled partitioning to determine the PARTITION limit key values for the table-controlled partitioning.

Ability to add partitions

You can use the ALTER TABLE statement to add a new partition to an existing partitioned table space and to each partitioned index in the table space. When you add a partition, DB2 uses the next physical partition that is not already in use until you reach the maximum number of partitions for the table space.

Example: Assume that a table space that contains a transaction table is divided into 5 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year. Table 2 shows a representation of the table space.

Table 2. Initial table space with 5 partitions

Partition	Limit value	Data set name for the partition
P001	12/31/2003	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001
P002	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/2007	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005

Assume that you want to add a new partition to handle the transactions for the next year. To add a partition, issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2008');
```

DB2 adds a new partition to the table space and to each partitioned index on the TRANS table. When the ALTER completes, you can use the new partition immediately. DB2 does not place the new partition in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

Ability to rotate partitions

Assume that the partition structure of the table space, as described in Table 2, is sufficient through the year 2008. When another partition is needed for the year 2009, you determine that the data for 2003 is no longer needed. You want to reuse the partition for the year 2003 to hold the transactions for the year 2009.

To rotate the first partition to be the last partition, issue the following statement:

```
ALTER TABLE TRANS ROTATE PARTITION FIRST TO LAST
   ENDING AT ('12/31/2009') RESET;
```

For a table with limit values in ascending order, the data in the ENDING AT clause must be higher than the limit value for previous partitions. DB2 chooses the FIRST

partition to be the partition with the lowest limit value. DB2 assigns the new limit value to P001 because it is the oldest partition (or the one with the lowest limit value). This partition holds all rows in the range between the new limit value of 12/31/2009 and the previous limit value of 12/31/2008.

The RESET keyword specifies that the existing data in the oldest partition is deleted. You can use the partition immediately after the ALTER completes. DB2 does not place the new partition in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

Table 3 shows a representation of the table space after the first partition is rotated to become the last partition.

Table 3. Rotating the low partition to the end

Partition	Limit value	Data set name for the partition
P002	12/31/2004	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A002
P003	12/31/2005	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A003
P004	12/31/2006	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A004
P005	12/31/2007	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A005
P006	12/31/2008	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A006
P001	12/31/2009	*catname*.DSNDB*x*.*dbname*.*psname*.I0001.A001

When you create your partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, use either ALTER TABLE ADD PARTITION to add partitions as needed, or, if rotating partitions is appropriate for your application, use ALTER TABLE ROTATE PARTITION to avoid adding another partition.

Ability to add columns to indexes

In Version 8 of DB2, you can append columns to the end of an existing index key with the ALTER INDEX statement.

If a column is added to a table and an index on that table in the same unit of work, the index is immediately available for access. However, if the column is added to the table and to the index in different units of work, DB2 puts the index in a REBUILD-pending (RBDP) state, and you need to run the REBUILD INDEX utility to make the index available.

If the index was created with DEFINE NO, and the underlying data sets have not yet been created, a restricted state is not set after columns are added to an index key.

Materialized query tables

DB2 UDB for z/OS now supports materialized query tables, which can simplify query processing and greatly improve the performance of dynamic SQL queries. Materialized query tables are particularly effective in data warehousing applications. A materialized query table contains information that is derived and summarized from other tables. Materialized query tables pre-calculate and store the results of queries that require expensive join and aggregation operations. DB2 uses automatic query rewrite to access data in a materialized query table.

If automatic query rewrite for materialized query tables is enabled, DB2 determines if a dynamic query or a portion of the query can be resolved by using a materialized query table. If so, DB2 rewrites the query to use the materialized query table instead of the underlying base tables to minimize query processing. Be aware that a materialized query table can yield query results that are not current if the base tables change after the materialized query table is updated.

To take advantage of using automatic query rewrite with materialized query tables, follow these steps:

1. Define materialized query tables. You can define materialized query tables using the CREATE TABLE or ALTER TABLE statements. The clauses DATA INITIALLY DEFERRED and REFRESH DEFERRED define a table as a materialized query table. You can define materialized query tables as MAINTAINED BY USER or MAINTAINED BY SYSTEM, which is the default.
2. Populate materialized query tables. Refresh materialized query tables periodically to maintain data currency with base tables. However, realize that refreshing materialized query tables can be an expensive process.
3. Enable automatic query rewrite for materialized query tables, and exploit its functions by submitting read-only dynamic queries. You can enable automatic query rewrite for materialized query tables by using the ENABLE QUERY OPTIMIZATION clause, which is the default in the CREATE TABLE statement. You can enable query rewrite for the dynamic queries by setting special registers CURRENT REFRESH AGE to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION to ALL, SYSTEM, or USER.
4. Evaluate the effectiveness of the materialized query tables. Drop under-utilized tables, and create new tables as necessary. You can use EXPLAIN to determine whether a materialized query table is used in a query.

The following example shows how DB2 can use a materialized query table to improve the performance of a simple query. Although most uses of materialized query tables will be much more complex, this example does illustrate some basic concepts.

Example: Suppose that you have a very large table named TRANS that contains one row for each transaction that a certain company processes. You want to tally the total amount of transactions by some time period. Although the table contains many columns, you are most interested in these four columns:

- YEAR, MONTH, DAY, which together identify the date of a transaction
- AMOUNT, which contains the amount of the transaction

To total the amount of all transactions between 1995 and 2000, by year, you would use the following query:

```
SELECT YEAR, SUM(AMOUNT)
  FROM TRANS
  WHERE YEAR >= '1995' AND YEAR <= '2000'
  GROUP BY YEAR
  ORDER BY YEAR;
```

This query might be very expensive to run, particularly if the TRANS table is a very large table with millions of rows and many columns.

Now suppose that you define a materialized query table named STRANS by using the following CREATE TABLE statement:

```
CREATE TABLE STRANS AS
  (SELECT YEAR AS SYEAR,
          MONTH AS SMONTH,
```

```
        DAY AS SDAY,
        SUM(AMOUNT) AS SSUM
    FROM TRANS
    GROUP BY YEAR, MONTH, DAY)
    DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

After you populate STRANS with a REFRESH TABLE statement, the table contains one row for each day of each month and year in the TRANS table.

Using the automatic query rewrite process, DB2 can rewrite the original query into a new query. The new query uses the materialized query table STRANS instead of the original base table TRANS:

```
SELECT SYEAR, SUM(SSUM)
  FROM STRANS
  WHERE SYEAR >= '1995' AND SYEAR <= '2000'
  GROUP BY SYEAR
  ORDER BY SYEAR
```

If you maintain data currency in the materialized query table STRANS, the rewritten query provides the same results as the original query. The rewritten query offers better response time and requires less CPU time.

Indexable predicates with mismatched data types

In previous releases of DB2, a predicate that compared a column to an expression was stage 1 and indexable only if the column and the expression had the same data type, and in many cases, the same length. These data type and length mismatches could cause performance problems that could not always be solved by changing application programs. For example, the application programmer cannot control whether the data types and lengths match in these situations:

- In most implementations of C and C++, there is no decimal data type, so host variables that are compared to columns with the DECIMAL data type must be defined with some other data types, such as float.
- Java™ does not have fixed-length data types. REXX does not have fixed-length string data types, except for the case when fixed-length strings are passed in an input SQLDA. For these languages, any comparisons between host variables and CHAR or GRAPHIC columns have a type mismatch.
- The programmer does not have access to the source code.

DB2 Version 8 makes changes that lessen the data type and length mismatch problem. Many predicates with mismatched data types and lengths are now stage 1 or indexable. Those predicates have the following general forms:

- *column op expression*
- *expression op column*
- *column* BETWEEN *expression1* AND *expression2*
- *column* IN (*list*)

In the preceding predicate types:

- *column* is a column of a table.
- *op* is one of the following comparison operators:
  - =
  - <
  - <=
  - >
  - >=
  - <>

- *expression* is an expression that contains any of the following elements:
  - Constants
  - Host variables
  - Special registers
  - Session variables
  - Parameter markers
  - Columns

  If the expression contains columns, and the other operands of the predicate also contain columns, no two columns can be in the same table.
- *list* meets all of the following conditions:
  - *list* contains only elements from the following list:
    - Constants
    - Host variables
    - Special registers
    - Session variables
    - Parameter markers
  - The predicate that contains *list* is not in the WHEN clause of a trigger.
  - For every element in *list*, *column=element* must be stage 1 and indexable.

If the predicate is of the form T1.*column op* T2.*column*, the join sequence determines which element is the column and which element is the expression. The inner table in the join sequence is considered to be the column, and the outer table of the join sequence is considered to be the expression.

For plans or packages that were created in a previous release of DB2, you need to rebind the plans or packages for static SQL statements to take advantage of this enhancement.

Predicates with one encoding scheme

Any of the previously-listed predicates are stage 1 and indexable, with the following restrictions:

- A numeric predicate is stage 1 but not indexable under the following conditions:
  - *op* is <>.
  - The *expression* is REAL or FLOAT, and *column* is DECIMAL with precision greater than 15.
- A string predicate is stage 1 but not indexable under the following conditions:
  - *op* is <>.
  - The *expression* is GRAPHIC or VARGRAPHIC, and *column* is CHAR or VARCHAR.

    An exception to this case is when *expression* is CHAR or VARCHAR and Unicode MIXED, and *op* is the equal (=) operator. In this case, the predicate is stage 1 and indexable.
  - *expression* and *column* are CHAR or VARCHAR, the length of *expression* is greater than the length of *column*, and *op* is not the equal (=) operator.
  - *expression* and *column* are GRAPHIC or VARGRAPHIC, the length of *expression* is greater than the length of *column*, and *op* is not the equal (=) operator.
  - *expression* is CHAR or VARCHAR, *column* is GRAPHIC or VARGRAPHIC, and *op* is not the equal (=) operator.
- A predicate in which *expression* is DATE, TIME, or TIMESTAMP, and *column* is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC is stage 2.

Predicates with more than one encoding scheme

Table 4 lists predicates that compare data in different encoding schemes and tells whether those predicates are indexable or stage 1. The following terms are used:

- U is a table in the Unicode encoding scheme, A is a table in the ASCII encoding scheme, and E is a table in the EBCDIC encoding scheme.
- *expression* is any expression that contains arithmetic operators, scalar functions, aggregate functions, concatenation operators, columns, constants, host variables, special registers, or date or time expressions.
- C2 *col expr* is an expression that includes any string column C2, as well as any of the any of the following elements:
  - Constants
  - Host variables
  - Special registers
  - Session variables
  - Parameter markers
- *op* is any of the operators =, <, <=, >, >=, or <>.
- *op-not-equal* is any of the operators <, <=, >, >=, or <>.

Table 4. Properties for string comparison predicates with more than one encoding scheme

Predicate type	Indexable?	Stage 1?
U.C1 *op* A.C2 U.C1 *op* E.C2 U.C1 *op* E.C2 *col expr* U.C1 *op* A.C2 *col expr* Conditions on these predicates: U is the inner table. The length of U.C1 is greater than or equal to the length of the other operand.	Y	Y
U.C1 = A.C2 U.C1 = E.C2 U.C1 = E.C2 *col expr* U.C1 = A.C2 *col expr* Conditions on these predicates: U is the inner table. The length of U.C1 is less than the length of the other operand.	Y	Y
U.C1 *op-not-equal* A.C2 U.C2 *op-not-equal* E.C2 U.C1 *op-not-equal* A.C2*col expr* U.C1 *op-not-equal* E.C2*col expr* Conditions on these predicates: U is the inner table. The length of U.C1 is less than the length of the other operand.	N	Y
A.C1 *op* E.C2 A.C1 *op* U.C2 A.C1 *op* E.C2 *col expr* A.C1 *op* U.C2 *col expr* Condition on these predicates: A is the inner table.	N	Y

Table 4. Properties for string comparison predicates with more than one encoding scheme (continued)

Predicate type	Indexable?	Stage 1?
E.C1 *op* A.C2 E.C1 *op* U.C2 E.C1 *op* A.C2 *col expr* E.C1 *op* U.C2 *col expr* Condition on these predicates: E is the inner table.	N	Y
E.C1 *op* A.C2*col expr* U.C1 *op* A.C2*col expr* Condition on these predicates: A is the inner table.	N	N
A.C1 *op* E.C2 *col expr* U.C1 *op* E.C2 *col expr* Condition on these predicates: E is the inner table.	N	N
A.C1 *op* U.C2 *col expr* E.C1 *op* U.C2 *col expr* Condition on these predicates: U is the inner table.	N	N

Index enhancements

In Version 8, DB2 makes a number of improvements to indexes:

- "Data-partitioned secondary indexes"
- "Backward index scan" on page 15
- "Varying-length index keys" on page 16
- "Longer index keys" on page 16
- "Distribution statistics" on page 16
- "Improved application availability for nonunique indexes" on page 17

Data-partitioned secondary indexes

A data-partitioned secondary index is a new type of partitioned index for Version 8 of DB2. For a data-partitioned secondary index, the number of index partitions equals the number of table space partitions. Index keys in partition *n* of the index reference only data in partition *n* of the table space. However, the data-partitioned secondary index is defined with different columns from the columns that define the table-controlled partitioning. Figure 2 on page 13 illustrates this concept.

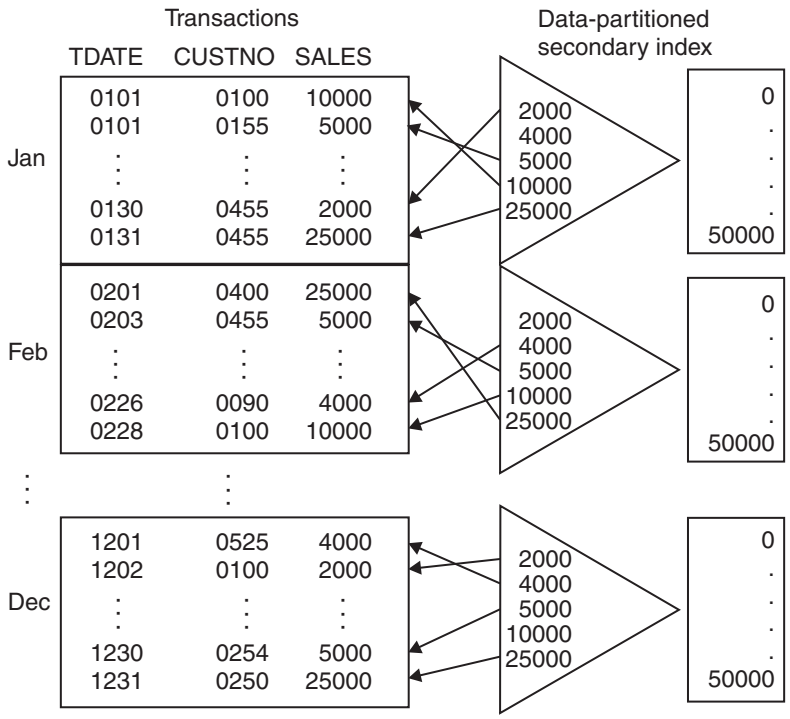

Figure 2. Example of a data-partitioned secondary index

In Figure 2, table TRANSACTIONS and index SALES_IX are defined like this:

```
CREATE TABLE TRANSACTIONS
 (TDATE CHAR(4),
  CUSTNO VARCHAR(4),
  SALES DECIMAL(9,0))
  IN TS1
  PARTITION BY (TDATE)
   (PART 1 ENDING AT ('0131'),
    PART 2 ENDING AT ('0228'),
    ...
    PART 12 ENDING AT ('1231'));

CREATE INDEX SALES_IX ON TRANSACTIONS (SALES) PARTITIONED;
```

Data-partitioned secondary index SALES_IX is defined on column SALES of the TRANSACTIONS table. However, the table and the index are physically partitioned by column TDATE. The result is that each partition of table TRANSACTIONS has data for only one month, and each partition of data-partitioned secondary index SALES_IX has keys for the SALES column values for the corresponding data partition of TRANSACTIONS.

Advantages of data-partitioned secondary indexes for utilities

This section describes the advantages that data-partitioned secondary indexes can provide over nonpartitioned secondary indexes for utility processing.

Partition-level utility operations can run on physical partitions: Because the keys for a given data partition reside in a single data-partitioned secondary index partition, utilities such as CHECK DATA, CHECK INDEX, COPY, REBUILD INDEX, RECOVER INDEX, REPAIR, and REPORT can operate on physical partitions, rather than logical partitions. The result can be greater availability.

Data-partitioned secondary indexes can make LOAD PART jobs run better: Data-partitioned secondary indexes can provide these advantages for running LOAD PART:

- When you run several LOAD PART jobs on different partitions of a partitioned table space, and the associated table has a nonpartitioned secondary index defined on it, contention can occur between the jobs because partitions can share pages of the index. However, if a data-partitioned secondary index is defined on the table space instead of a nonpartitioned secondary index, the partitions do not share pages of the index, so contention is reduced.
- During parallel LOAD PART execution with a data-partitioned secondary index, LOAD inserts the data-partitioned secondary index keys into a separate index structure for each partition, in key order. This is more efficient than key insertion for a nonpartitioned secondary index.

Data-partitioned secondary indexes can reduce data sharing overhead: Data sharing users sometimes do batch processing of partitions in a partitioned table space in parallel, with each batch job processing one or more partitions. This is done to isolate work for data sharing members to specific partitions of a table space to alleviate contention. This technique can reduce intersystem read-write interest in physical partitions, which reduces data sharing overhead. However, if nonpartitioned secondary indexes are defined on the table space, running batch jobs in parallel is less effective because contention occurs on the indexes. Using data-partitioned secondary indexes instead can alleviate this problem because different data sharing members can operate on different index partitions, as well as different table space partitions.

Data-partitioned secondary indexes can eliminate the BUILD2 phase of REORG TABLESPACE: The BUILD2 phase corrects nonpartitioning indexes for REORG TABLESPACE PART SHRLEVEL REFERENCE or CHANGE. If you define only partitioned indexes on a table, you do not need the BUILD2 phase. Therefore, if you define a data-partitioned secondary index on a table that serves the same purpose as a nonpartitioned index served in previous releases of DB2, you can avoid the BUILD2 phase.

Data-partitioned secondary indexes provide more efficient index backup and recovery: You can copy and recover data-partitioned secondary indexes by partition. In addition, you can rebuild individual partitions of a data-partitioned secondary index in parallel for a faster rebuild of the entire index.

Advantages of data-partitioned secondary indexes for queries

A data-partitioned secondary index can provide a performance advantage for a query that meets the following criteria:

- The query has predicates that contain columns that are in the data-partitioned secondary index.
- The query contains additional predicates on the partitioning columns of the table that limit the result table to a subset of the partitions in the table.

Example: Suppose that, in addition to the SALES_IX data-partitioned secondary index, the TRANSACTIONS table also has the partitioned index TDATE_IX, which is defined like this:

```
CREATE INDEX TDATE_IX ON TRANSACTIONS (TDATE) PARTITIONED CLUSTER;
```

The following query on the TRANSACTIONS table can use data-partitioned secondary index SALES_IX for better performance:

```
SELECT CUSTNO, SALES
 FROM TRANSACTIONS
 WHERE TDATE BETWEEN '0101' AND '0228' AND
 SALES >= 10000;
```

This query meets the criteria for making efficient use of a data-partitioned secondary index:

- The predicate includes the SALES column, which is in the data-partitioned secondary index.
- The predicate includes the partitioning key TDATE, which limits the selected data to only the first two partitions of the table.

Disadvantages of data-partitioned secondary indexes for queries

A data-partitioned secondary index is not always appropriate for queries. Among the reasons are:

- For queries that do not include the partitioning columns, using a data-partitioned secondary index means that DB2 must do an index scan for each partition.
- A data-partitioned secondary index cannot be a unique index, so it cannot be used to enforce uniqueness across partitions.

Backward index scan

Version 8 of DB2 includes the capability for backward index scan. Backward index scan can improve performance of a SELECT statement with an ORDER BY *column* DESC clause because it reduces the need for DB2 to do sorts. In addition, the backward index scan capability can reduce the need for descending indexes because DB2 can use ascending indexes to scan backward.

DB2 can use an index for a backward scan if the following conditions are true:

- The index is defined on the same columns as the columns in the ORDER BY clause, or the index is defined on the same columns as the columns in the ORDER BY clause, followed by other columns.
- For each column that is in the ORDER BY clause, the ordering that is specified in the index is the opposite of the ordering that is specified in the ORDER BY clause.

Example: Suppose that index ACCT_STAT_IX is defined like this:

```
CREATE INDEX ACCT_STAT_IX
 ON ACCT_STAT
  (ACCT_NUM ASC,
   STATUS_DATE ASC,
   STATUS_TIME DESC);
```

Now suppose that you want to fetch rows using a cursor that is declared like this:

```
DECLARE CURSOR C1 SENSITIVE STATIC SCROLL FOR
 SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
 ORDER BY ACCT_NUM DESC, STATUS_DATE DESC, STATUS_TIME ASC;
```

Because ACCT_NUM and STATUS_DATE are in ascending order in ACCT_STAT_IX and descending order in the ORDER BY clause, and STATUS_TIME is in descending order in ACCT_STAT_IX and ascending order in the ORDER BY clause, DB2 can use ACCT_STAT_IX to do a backward scan without doing a sort.

For plans or packages that were created in a previous release of DB2, you need to rebind the plans or packages for static SQL statements to take advantage of this enhancement.

Varying-length index keys

In previous releases of DB2, VARCHAR and VARGRAPHIC columns in indexes were padded to the maximum lengths of the columns. In Version 8 of DB2, index keys for varying-length columns can be varying-length. Varying-length keys have the following advantages:

- Varying-length key columns usually result in smaller indexes because the index keys use less than the maximum number of bytes that are defined for the columns.
- Varying-length keys can use index-only access to the data. Fixed-length keys for varying-length columns cannot do this.
- Other DB2 UDB family members have varying-length index keys, so providing this capability increases compatibility with the DB2 UDB family.

Indexes in which VARCHAR or VARGRAPHIC columns are padded to their maximum length are PADDED indexes. Indexes that are not padded to their maximum length are NOT PADDED indexes. You can specify whether an index is PADDED or NOT PADDED by specifying the PADDED or NOT PADDED keyword in CREATE INDEX or ALTER index. You can also set the default index padding mode through the new DEFIXPD subsystem parameter.

Indexes that were created in previous releases of DB2 are PADDED.

If you alter an index that contains VARCHAR or VARGRAPHIC columns from PADDED to NOT PADDED (or from PADDED to NOT PADDED), DB2 places that index in a restricted REBUILD-pending (RBDP) state. You need to run the REBUILD INDEX, REORG TABLESPACE, or LOAD REPLACE utility to reset the RBDP state.

Longer index keys

Version 8 of DB2 increases the maximum length of an index key from 255 bytes to 2000 bytes. The increased key length has the following advantages:

- More compatibility with the DB2 UDB family
- Simplified conversion of vendor applications from EBCDIC or ASCII to Unicode

  For example, a table column that is CHAR(10) might be represented as GRAPHIC(20) in Unicode. An index on that column also requires twice as many bytes. In cases like this, the previous maximum key length might be inadequate.

Distribution statistics

To run efficiently, data warehousing, data mining, and ad hoc query applications need statistics on columns that are in predicates, regardless of whether they are leading columns of an index. In addition, distribution statistics on non-leading index columns or non-indexed columns let DB2 make better access path decisions when data is asymmetrically distributed.

In Version 8 of DB2, you can use RUNSTATS to collect the following additional statistics:

- Frequency distributions for non-indexed columns or groups of columns
- Cardinality values for groups of non-indexed columns

- Least-frequently occurring values, most-frequently occurring values, or both, for any group of columns

Example: Collecting cardinality statistics for a column group: Run RUNSTATS with the COLGROUP parameter to collect cardinality statistics on a column group that consists of columns EDLEVEL, JOB, and SALARY of the employee table.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
 TABLE(DSN8810.EMP)
 COLGROUP(EDLEVEL, JOB, SALARY)
```

Example: Collecting most-frequent and least-frequent value statistics for a column group: Run RUNSTATS with the COLGROUP and FREQVAL BOTH parameters to collect cardinality statistics and statistics on the 15 most-frequent and least-frequent values for a column group that consists of columns EDLEVEL, JOB, and SALARY of the employee table.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
 TABLE(DSN8810.EMP)
 COLGROUP(EDLEVEL, JOB, SALARY) FREQVAL COUNT 15 BOTH
```

Improved application availability for nonunique indexes

For improved application availability, INSERT, UPDATE, and DELETE operations can occur on a table with a nonunique index that is in REBUILD-pending status.

Reoptimizing the access path at run time

Version 8 of DB2 UDB for z/OS introduces the following bind options for reoptimizing the access path at run time:

REOPT(ALWAYS) DB2 determines and caches the access path for any SQL statement with variable values each time the statement is run. REOPT(ALWAYS) replaces the REOPT(VARS) option from previous versions of DB2.

REOPT(ONCE) DB2 determines and caches the access path for any SQL statement with variable values only once at run time, using the first set of input variable values. If the statement is run multiple times, DB2 does not reoptimize each time. The REOPT(ONCE) bind option works only with dynamic SQL statements, and it allows DB2 to store the access path for dynamic SQL statements in the dynamic statement cache.

REOPT(NONE) DB2 determines the access path at bind time, and does not change the access path at run time.

Performance enhancements for star join

The information under this heading, up to "Cost-based parallel sorting" on page 19, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 149.

In Version 8, DB2 UDB for z/OS introduces three performance enhancements for star joins: sparse indexing[1], a dedicated virtual memory pool, and avoidance of snowflake materialization. This section discusses sparse indexing and the dedicated virtual memory pool.

Sparse indexing: Sparse indexing for star joins can significantly improve the performance of data warehousing applications. Because many data warehousing applications rely on the highly normalized structure of star schema design, these applications can have a large number of snowflake work files. Before Version 8, DB2 could not use indexes for snowflake work files. Instead of using indexes, DB2 tended to join snowflake work files by using a costly sort-merge join or nested-loop join with table space scan on snowflake workfiles. In Version 8, sparse indexes for star joins provide DB2 with more efficient access paths.

DB2 Version 8 can choose a sparse-index access path if an equal join predicate exists between the fact table and each dimension table (snowflake composite).

Sparse indexing for star joins improves performance in data warehousing applications by enabling the following efficient actions:

- Avoiding the sort-merge join or nested-loop join with table space scan on snowflake workfiles. The sparse-index access path can be a particularly important performance enhancement when it eliminates single or multiple large composite sorts. The sparse index join method can also reduce parallelism overhead.
- Expediting the skipping of unqualified keys. The increased efficiency results in a significant I/O reduction. The CPU cost reduction only becomes significant for a large sort or for multiple sorts.
- Increasing the exploitation of parallelism.

When DB2 chooses the sparse-index access path, the ACCESS_TYPE column in the PLAN_TABLE contains the character T for the work file.

Dedicated virtual memory pool: In DB2 Version 8, you can create a dedicated virtual memory pool for star join operations. When the virtual memory pool is enabled for star joins, DB2 caches data from workfiles that are used by star join queries. A virtual memory pool dedicated to star join operations has the following advantages:

- Immediate data availability. During a star join operation, workfiles might be scanned many times. If the workfile data is cached in the dedicated virtual memory pool, that data is immediately available for join operations.
- Reduced buffer pool contention. Because the dedicated virtual memory pool caches data separately from the workfile buffer pool, contention with the buffer pool is reduced. Reduced contention improves performance particularly when sort operations are performed concurrently.

For information about determining the size of your dedicated virtual memory pool and implementing your dedicated virtual memory pool, see *DB2 Administration Guide*.

1. APAR PQ614588 adds support for sparse indexing for star joins in Version 7 of DB2 UDB for z/OS and OS/390.

Cost-based parallel sorting

Before Version 8 of DB2, the number of tables in a sort determined whether DB2 used a parallel sort. Single-table sorts used parallel sorting; multiple-table sorts did not. In Version 8, DB2 UDB for z/OS introduces cost-based parallel sorting. In Version 8, DB2 determines whether to use a parallel or non-parallel sort based on cost considerations, including sort data size and parallel degrees. You can determine whether a sort is executed in parallel by using EXPLAIN.

Visual Explain enhancements

Visual Explain for DB2 UDB for z/OS is a workstation tool that provides graphical depictions of the access plans that DB2 chooses for your SQL queries and statements. Such graphs eliminate the need to manually interpret plan table output. The relationships between database objects, such as tables and indexes, and operations, such as table space scans and sorts, are clearly illustrated in the graphs. You can choose to have the attributes for these objects and operations displayed next to the graph. Figure 3 shows an example of such a graph. In this graph, all of the attributes for the sort operation are displayed on the left, because the SORT node is highlighted on the right. Notice the navigation tree above the attribute list. You can use this tree to link to related objects.

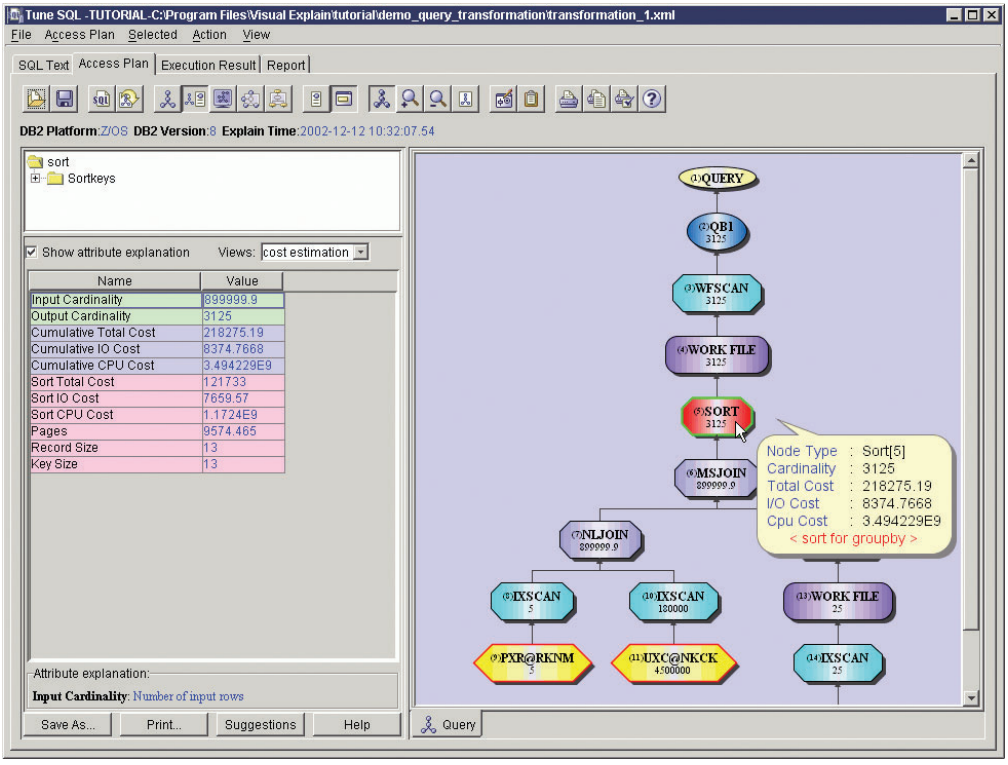

Figure 3. Example of an access plan graph in Visual Explain

You can also use Visual Explain to generate customized reports on explainable statements, to view subsystem parameters, and to view data from the plan table, the statement table, and the function table.

This release of Visual Explain also includes the following enhancements:

- More context-sensitive tuning suggestions are provided. You can link to these suggestions directly from the graph.

- You can link from the graph to attributes and descriptions for each object or operation that is used in the access plan.
- Each graph can display either one or multiple query blocks, so that you can view the entire access plan in one graph. In previous versions of Visual Explain, each graph displayed only one query block.
- You can use Visual Explain to catalog and uncatalog databases on your local machine.
- You can use Visual Explain to run a query and view the formatted results.

64-bit virtual storage

Version 8 of DB2 UDB for z/OS, through deep integration with the IBM zSeries® 800, 900, 990, or equivalent, now supports 64-bit virtual storage in the DBM1 address space. DB2 previously supported 31-bit virtual storage, which afforded DB2 a 2-GB address space. By supporting 64-bit virtual storage, DB2 UDB for z/OS now supports a 16-exabyte address space. Because the 16-exabyte address space is 8 billion times larger than the 2-GB address space, it can provide significant virtual storage constraint relief.

The introduction of 64-bit virtual storage in Version 8 of DB2 UDB for z/OS brings a reorganization of the address space. Many entities are now stored in the new virtual storage area above the former 2-GB limit. Entities that are stored above that limit are said to be ″above the bar.″ The entities that move above the bar in Version 8, and the advantages related to each move, are shown in Table 5.

Table 5. Entities above the 2-GB bar and related advantages

Entity above the bar	Advantage
Buffer pools	Larger buffer pools reduce I/O for random access and enable larger page sizes, which benefit sequential access. Buffer pool management becomes easier, and hiperpools and data spaces are eliminated.
DBDs in the EDM pool	More database objects can be defined, which can reduce I/O to and from the DB2 directory.
Compression dictionaries	Open and close operations can be avoided when a large number of compressed table spaces must be accessed.
Global dynamic statement cache	A larger cache avoids dynamic rebinds.
IRLM locks	More storage for IRLM locks enables smaller lock granularity, which results in reduced lock contention.
Sort pools	Frees up space below the bar for other uses.
RIDLISTs for the RID pool	Frees up space below the bar for other uses.

Although 64-bit virtual storage changes buffer pool management, you can run your existing applications and use your existing buffer pools when you migrate to Version 8 of DB2 UDB for z/OS. The buffer pool names (BP0, BP1, and so forth) do not change. The page size options remain 4 KB, 8 KB, 16 KB, and 32 KB. When you migrate, DB2 determines the buffer pool size based on the following equation:

```
VPSIZE + HPIZE = BPSIZE
```

VPSIZE is the old virtual pool size, HPSIZE is the old virtual pool size, and BPSIZE is the new buffer pool size. When you install DB2 UDB for z/OS as a new Version 8 subsystem, you can specify the buffer pool sizes during installation.

Data sharing enhancements

The following data sharing enhancements can improve your performance and availability:

- "Improved LPL recovery"
- "Reduction of locking overhead for data sharing workloads"
- "Reduction of buffer management overhead costs for data sharing workloads" on page 22
- "Improved index split performance for data sharing" on page 22
- "Resolution of indoubt units of recovery in restart light" on page 22

Improved LPL recovery

With Version 8 of DB2, only pages in the logical page list (LPL) are locked as part of the recovery process, leaving the remaining pages in the page set or partition accessible to DB2 applications while LPL recovery is in progress. This improves system performance and enhances data availability.

Prior to Version 8, you needed to manually recover pages that DB2 put into the LPL. In Version 8, DB2 provides support for automatic recovery of LPL pages. When pages are added to the LPL, DB2 issues message DSNB250E, which is enhanced to indicate the reason the pages are added to the LPL. DB2 then attempts automatic recovery, except in the following situations:

- Disk I/O errors
- During DB2 restart or end_restart times
- Group buffer pool structure failures
- 100% loss of connection to the group buffer pool

If automatic-LPL recovery completes successfully, DB2 deletes the pages from the LPL and issues message DSNI021I, which indicates completion.

Reduction of locking overhead for data sharing workloads

Version 8 contains several locking enhancements to improve locking performance in a data sharing environment:

- The IX and IS mode parent L-locks of different members no longer encounter global lock contention because both locks are now treated as S-type locks in XES and in the coupling facility.

  The reduction of global lock contention improves performance, especially for plans and packages that are bound with RELEASE(COMMIT), and greatly reduces the need to use the RELEASE(DEALLOCATE) bind option in conjunction with thread reuse to obtain good performance.
- DB2 now uses the parent P-lock (at the page set or partition level) instead of the parent L-lock to determine whether it is necessary to propagate child locks to the coupling facility.

  Use of the parent P-lock increases availability because retained parent L-locks no longer block access to an entire table space or partition when a member fails. Use of the parent P-lock also improves performance because child lock propagation is not an issue as inter-DB2 interest changes on parent L-locks.

As a result of these changes it may be necessary to increase the size of the lock structure in the coupling facility. These enhancements are available only in new-function mode, and only after all members of the data sharing group have been stopped without error and restarted.

Reduction of buffer management overhead costs for data sharing workloads

Version 8 makes use of two new batch processes to reduce the amount of traffic to and from the coupling facility when you are running z/OS Version 1 Release 4 and coupling facility level 12. DB2 can now write and register multiple pages to a group buffer pool when you use the new z/OS Write And Register Multiple (WARM) command. And when you use a single Read For CastOut Multiple (RFCOM) command, DB2 can read multiple pages from a group buffer pool for castout processing.

You can expect the greatest performance benefits for data sharing workloads that update a large number of pages belonging to group buffer pool dependent objects.

Improved index split performance for data sharing

Version 8 greatly reduces the number of log and coupling facility operations that are associated with an index page split. In previous versions of DB2, index page splits require up to five separate writes to the group buffer pool and emptying of the DB2 log buffers. With Version 8, index page splits are optimized, improving performance for high-volume INSERT OLTP workloads and other operations.

Resolution of indoubt units of recovery in restart light

In versions of DB2 before Version 8, starting a DB2 member in LIGHT(YES) mode (restart light) removes retained locks with minimal disruption in the event of a system failure. Restart light is improved in Version 8. If indoubt units of recovery (URs) exist at the end of restart recovery, DB2 remains running so that the indoubt URs can be resolved. After all the indoubt URs are resolved, the DB2 member that is running in LIGHT(YES) mode shuts down and can be restarted normally.

Improved space allocation

Version 8 of DB2 introduces improved default primary and secondary space allocations for DB2-managed data sets. Beginning in Version 8, the DB2-supplied default values for table space and index space allocation size are increased. Default allocations are in cylinders now, which can result in better performance of mass inserts, prefetch operations, and the LOAD, REORG, and RECOVER utilities. Additionally, by improving the method for allocating secondary extents, the likelihood of out-of-extents errors is decreased.

The objectives of the Version 8 space allocation enhancements are:

- To improve performance, increase data availability, and limit the occurrence of outages caused by lack of space
- To prevent a DB2-managed data set from reaching the VSAM maximum extent limit of 255 before it reaches the maximum page set size
- To eliminate the need to specify primary and secondary quantity values for DB2-managed data sets when creating or modifying table spaces and indexes

The new space allocation methods affect both new DB2-managed data sets and existing data sets that require additional extents. See “Migration considerations” on page 67 for an understanding of how space allocation changes for DB2-managed data sets affect your site

New default primary space allocation value

By default, DB2 now uses the following values for primary space allocation of DB2-managed data sets:

- 1 cylinder (720 KB) for non-LOB table spaces
- 10 cylinders for LOB table spaces
- 1 cylinder for indexes

To indicate that you want DB2 to use the default values for primary space allocation of table spaces and indexes, specify a value of 0 for the following parameters on installation panel DSNTIP7, as shown in Table 6.

Table 6. DSNTIP7 parameter values for managing space allocations

Installation panel DSNTIP7 parameter	Recommended value
TABLE SPACE ALLOCATION	0
INDEX SPACE ALLOCATION	0

Thereafter:

- On CREATE TABLESPACE and CREATE INDEX statements, do not specify a value for the PRIQTY option.
- On ALTER TABLESPACE and ALTER INDEX statements, specify a value of -1 for the PRIQTY option.

DB2 stores a value of -1 in the PQTY column of either the SYSIBM.SYSTABLEPART or SYSIBM.SYSINDEXPART table when it uses the default value for primary space allocation.

Primary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

For those situations in which the default primary quantity value is not large enough, you can specify a larger value for the PRIQTY option when creating or altering table spaces and indexes. DB2 always uses a PRIQTY value if one is explicitly specified.

If you want to prevent DB2 from using the default value for primary space allocation of table spaces and indexes, specify a non-zero value for the TABLE SPACE ALLOCATION and INDEX SPACE ALLOCATION parameters on installation panel DSNTIP7.

New sliding scale for secondary space allocation

DB2 can now calculate the amount of space to allocate to secondary extents by using a sliding scale algorithm. The first 127 extents are allocated in increasing size, and the remaining extents are allocated based on the initial size of the data set:

- For 32 GB and 64 GB data sets, each extent is allocated with a size of 559 cylinders.
- For data sets that range in size from less than 1 GB to 16 GB, each extent is allocated with a size of 127 cylinders.

This approach has several advantages:

- It minimizes the potential for wasted space by increasing the size of secondary extents slowly at first.

- It prevents very large allocations for the remaining extents, which would likely cause fragmentation.
- It does not require users to specify SECQTY values when creating and altering table spaces and index spaces.
- It is theoretically possible to always reach maximum data set size without running out of secondary extents.

DB2 stores a value of -1 in the SQTY column of the SYSIBM.SYSTABLEPART or SYSIBM.SYSINDEXPART table when it uses the default value for secondary space allocation.

Maximum allocation is shown in Table 7. This table assumes that the initial extent that is allocated is one cylinder in size.

Table 7. Maximum allocation of secondary extents

Maximum data set size, in GB	Maximum allocation, in cylinders	Extents required to reach full size
1	127	54
2	127	75
4	127	107
8	127	154
16	127	246
32	559	172
64	559	255

DB2 uses a sliding scale for secondary extent allocations of table spaces and indexes when:

- You do not specify a value for the SECQTY option of a CREATE TABLESPACE or CREATE INDEX statement.
- You specify a value of -1 for the SECQTY option of an ALTER TABLESPACE or ALTER INDEX statement

Otherwise, DB2 always uses a SECQTY value for secondary extent allocations, if one is explicitly specified.

Exception: For those situations in which the calculated secondary quantity value is not large enough, you can specify a larger value for the SECQTY option when creating or altering table spaces and indexes. However, in the case where the OPTIMIZE EXTENT SIZING parameter is set to YES and you specify a value for the SECQTY option, DB2 uses the value of the SECQTY option to allocate a secondary extent only if the value of the option is larger than the value that is derived from the sliding scale algorithm. The calculation that DB2 uses to make this determination is:

```
Actual secondary extent size = max ( min ( ss_extent, MaxAlloc ), SECQTY )
```

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size. This approach allows you to reach the maximum page set size faster. Otherwise, DB2 uses the value that is derived from the sliding scale algorithm.

If you do not provide a value for the secondary space allocation quantity, DB2 calculates a secondary space allocation value equal to 10% of the primary space allocation value and subject to the following conditions:

- The value cannot be less than 127 cylinders for data sets that range in initial size from less than 1 GB to 16 GB, and cannot be less than 559 cylinders for 32 GB and 64 GB data sets.
- The value cannot be more than the value that is derived from the sliding scale algorithm.

The calculation that DB2 uses for the secondary space allocation value is:

```
Actual secondary extent size = max ( 0.1 × PRIQTY, min ( ss_extent, MaxAlloc ) )
```

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size.

Secondary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

If you do not want DB2 to extend a data set, you can specify a value of 0 for the SECQTY option. Specifying 0 is a useful way to prevent DSNDB07 work files from growing out of proportion.

If you want to prevent DB2 from using the sliding scale for secondary extent allocations of table spaces and indexes, specify a value of NO for the OPTIMIZE EXTENT SIZING parameter on installation panel DSNTIP7.

Secondary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

More options for data security in TCP/IP networks

Version 8 of DB2 introduces two new TCP/IP security mechanisms:

- A more secure mechanism for verifying a remote client's port of entry
- Improved encrypted security mechanisms

More secure mechanism for verifying a remote client's port of entry

When a remote TCP/IP client attempts to establish a connection to a DB2 UDB for z/OS server, the user ID that is associated with the incoming request is subjected to RACF verification. Currently, if the RACF APPCPORT class is active, RACF also verifies that the user ID is authorized to access z/OS from the client's port of entry. Beginning in this release, DB2 can provide a SERVAUTH profile name to RACF when verifying the port of entry of a user ID.

This improved security mechanism is dependent on the use of the following features in Version 1 Release 5 of z/OS:

- The NETACCESS statement

  Use the NETACCESS statement to configure network access control use of z/OS Communications Server. This allows DB2 to restrict the access of particular users from specific IP networks. See *z/OS Communications Server: IP Configuration Guide* for complete information about using the NETACCESS statement.
- The SERVAUTH class resource

Use the SERVAUTH class resource in RACF to protect the network security zones, as defined by the NETACCESS statement. See *z/OS Security Server RACF Security Administrator's Guide* for complete information about using the SERVAUTH class resource.

In prior z/OS releases, the port of entry that was used in the RACROUTE VERIFY call was the literal string 'TCPIP'. Beginning in Version 1 Release 5 of z/OS, if TCP/IP network access control is configured and the RACF SERVAUTH class is active, the port of entry that is used in the RACROUTE VERIFY call is the security zone name of the port of entry for the remote client.

See *DB2 Administration Guide* for detailed instructions on using the RACF SERVAUTH class and TCP/IP network access control.

Improved encrypted security mechanisms

New Distributed Relational Database Architecture™ (DRDA®) security options provide the following data security improvements in distributed computing environments:

- DB2 UDB for z/OS servers can provide secure, high-speed data encryption and decryption.
- DB2 UDB for z/OS requesters now have the option of encrypting user IDs and, optionally, passwords when they connect to remote servers. Requesters can also encrypt security-sensitive data when communicating with servers, so that the data is secure when traveling over the network.

By default, encrypted security mechanisms use the z/OS integrated cryptographic service facility (ICSF). ICSF is a software element of z/OS that works with a required hardware cryptographic feature and RACF (or equivalent) to provide secure, high-speed cryptographic services. ICSF supports cryptography by the IBM Common Cryptographic Architecture (CCA), which is based on the DES algorithm. See *Integrated Cryptographic Service Facility Administrator's Guide* for more detailed information about ICSF. If ICSF is not available, is not installed or configured properly, or is not active, DB2 uses the existing BSAFE services for only those security mechanisms that are supported by DB2 UDB for z/OS servers in previous releases.

Authentication mechanisms used by DB2 UDB for z/OS as a server

As a server, DB2 UDB for z/OS can accept either SNA or DRDA authentication mechanisms. Therefore, DB2 can authenticate remote users from either the security tokens that are obtained from the SNA ATTACH (FMH-5) or from the DRDA security commands that are described by each of the protocols. If TCP/IP protocols are used, the following additional authentication methods are now supported:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data
- Encrypted user ID, encrypted password, encrypted new password, and encrypted security-sensitive data

Prerequisite: ICSF must be installed, configured, and active before servers can offer encryption and decryption services.

Authentication mechanisms used by DB2 UDB for z/OS as a requester

As a requester, DB2 UDB for z/OS chooses SNA or DRDA security mechanisms based on the network protocol and the authentication mechanisms you use. If you use TCP/IP protocols, the following additional DRDA authentication mechanisms are now supported:

- Encrypted user ID and encrypted password
- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

Prerequisite: ICSF must be installed, configured, and active before requesters can use the new encryption options.

For performance reasons, the entire network stream is not encrypted. Only the following security-sensitive types of data are encrypted:

- SQL statements that are being prepared, executed, or bound to an RDB package.
- SQL statement variable descriptions that appear in an SQL statement.
- SQL statement attributes that are being prepared.
- SQL program variable data that consists of input data to an SQL statement during an open or execute operation. This also includes a description of the data.
- SQL reply data that consists of output data from the processing of a SQL statement. This also includes a description of the data.
- Query answer set data that consists of the answer set that results from a query.
- SQL result set reply data and SQL result set column information reply data.
- Input or output LOB data.
- A description of the data that is returned from the server as the result of a describe operation.

Changes to the communications database

The SECURITY_OUT column of the SYSIBM.IPNAMES table now supports two new DRDA security options:

D The option is ″userid and security-sensitive data encryption″. Outbound connection requests contain an authorization ID and no password. The authorization ID used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value of the USERNAMES column.

E The option is ″userid, password, and security-sensitive data encryption″. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. The USERNAMES column must specify 'O'.

In addition, the security option 'P' now supports encryption:

P The option is ″password″. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. The USERNAMES column must specify 'O'. This option indicates that the user ID and the password are to be encrypted, if the server supports encryption. Otherwise, the user ID and the password are sent to the partner in clear text.

System-level point-in-time recovery

Version 8 provides an enhanced system-level point-in-time recovery capability. You can make fast volume-level backups of a DB2 subsystem or data-sharing group with minimal disruption and recover a subsystem or data-sharing group to any point in time, regardless of whether you have uncommitted units of work.

The new BACKUP SYSTEM utility takes fast volume-level copies of DB2 databases and logs with minimal disruption. You can copy both the data and logs or only the data. Previously, to make a system-level backup, you needed to issue the SET LOG SUSPEND command, which stops logging and thus prevents any new database updates. A BACKUP SYSTEM job does not stop logging; it needs only to wait for the following events to complete:

- 32-KB page writes
- Read-only switching
- Data set extensions

The BACKUP SYSTEM utility can operate on an entire data-sharing group, whereas the SET LOG SUSPEND command must be issued for each data-sharing member.

The new RESTORE SYSTEM utility recovers a DB2 subsystem to an arbitrary point in time. This utility automatically handles any creates, drops, and LOG NO events that might have occurred between the time the backup was taken and the recovery point in time.

The BACKUP SYSTEM and RESTORE SYSTEM utilities rely on new DFSMShsm™ services in z/OS V1R5 that automatically monitor which volumes need to be copied. The BACKUP SYSTEM and RESTORE SYSTEM utilities use copy pools, which are new constructs in z/OS DFSMShsm V1R5. A *copy pool* is a construct that contains the names of SMS-managed storage groups that can be backed up and restored with a single command. These storage groups are also referred to as the source storage groups. Each of these source storage groups contains the name of an associated copy-pool backup storage group, which contains eligible volumes for the backups. Each DB2 subsystem can have up to two copy pools, one for databases and one for logs. BACKUP SYSTEM copies the volumes that are associated with these copy pools at the time of the copy.

To use the BACKUP SYSTEM and RESTORE SYSTEM utilities, you must ensure that the following conditions are true:

- The data sets that you want to copy are SMS-managed data sets.
- You are running z/OS V1R5 or above.
- You have disk control units that support ESS FlashCopy®.
- You have defined a copy pool for your database data. If you plan to also copy the logs, define another copy pool for your logs. Use the DB2 naming convention for both of these copy pools.
- You have defined an SMS backup storage group for each storage group in the copy pools.

Additional parameters

This section contains information about new subsystem parameters and about subsystem parameters that have been changed to be dynamically updatable in Version 8. See *DB2 Installation Guide* for complete details about these parameters.

New subsystem parameters

Several subsystem parameters have been added to installation panels (see Table 8). As a result, the values you choose for these parameters are used during migration to a new release of DB2. All new parameters are dynamically updateable.

Table 8. Parameters that have been added to installation panels

Subsystem parameter	Panel	Field name
ACCUMACC	DSNTIPN	DDF/RRSAF ACCUM
ACCUMUID	DSNTIPN	AGGREGATION FIELDS
AEXITLIM	DSNTIPP	AUTH EXIT LIMIT
DSVCI	DSNTIP7	VARY DS CONTROL INTERVAL
EDMDBDC	DSNTIPC	EDM DBD CACHE
EDMSTMTC	DSNTIPC	EDM STATEMENT CACHE
IXQTY	DSNTIP7	INDEX SPACE DEFAULT SIZE
LRDRTHLD	DSNTIPE	LONG-RUNNING READER
MAINTYPE	DSNTIP4	CURRENT MAINT TYPES
MAX_NUM_CUR	DSNTIPX	MAX OPEN CURSORS
MAX_ST_PROC	DSNTIPX	MAX STORED PROCS
MGEXTSZ	DSNTIP7	OPTIMIZE EXTENT SIZING
NEWFUN	DSNTIPA1	INSTALL TYPE
PADIX	DSNTIPE	PAD INDEXES BY DEFAULT
PADNTSTR	DSNTIP4	PAD NUL-TERMINATED
REFSHAGE	DSNTIP4	CURRENT REFRESH AGE
SJMXPOOL	DSNTIP8	STAR JOIN MAX POOL
SKIPUNCI	DSNTIP8	SKIP UNCOMM INSERTS
SMF89	DSNTIPN	USAGE PRICING
STARJOIN	DSNTIP8	STAR JOIN QUERIES
SVOLARC	DSNTIPA	SINGLE VOLUME
TSQTY	DSNTIP7	TABLE SPACE DEFAULT SIZE
UIFCIDS	DSNTIPN	UNICODE IFCIDS
VOLTDEVT	DSNTIPA2	TEMPORARY UNIT NAME

Subsystem parameters changed to dynamically updatable

Several subsystem parameters have been changed to be dynamically updatable as shown in Table 9. You can change these values by using the SET SYSPARM command to load the new module.

Table 9. Subsystem parameters that can now be dynamically updated

Subsystem parameter	Panel	Field name
CACHEDYN	DSNTIP4	CACHE DYNAMIC SQL
CHGDC	DSNTIPO	DPROP SUPPORT
EDPROP	DSNTIPO	DPROP SUPPORT
EXTRAREQ	DSNTIP5	EXTRA BLOCKS REQ

Table 9. Subsystem parameters that can now be dynamically updated (continued)

Subsystem parameter	Panel	Field name
EXTRASRV	DSNTIP5	EXTRA BLOCKS SRV
IDTHTOIN	DSNTIPR	IDLE THREAD TIMEOUT
IMMEDWRI	DSNTIP4	IMMEDIATE WRITE
MAXKEEPD	DSNTIPE	MAX KEPT DYN STMTS
MAXTYPE1	DSNTIPR	MAX TYPE 1 INACTIVE
PARTKEYU	DSNTIP4	UPDATE PART KEY COLS
POOLINAC	DSNTIP5	POOL THREAD TIMEOUT
RESYNC	DSNTIPR	RESYNC INTERVAL
SRTPOOL	DSNTIPC	SORT POOL SIZE
SYSADM	DSNTIPP	SYSTEM ADMIN 1
SYSADM1	DSNTIPP	SYSTEM ADMIN 2
SYSOPR1	DSNTIPP	SYSTEM OPERATOR 1
SYSOPR2	DSNTIPP	SYSTEM OPERATOR 2
TCPALVER	DSNTIP5	TCP/IP ALREADY VERIFIED
TCPKPALV	DSNTIP5	TCP/IP KEEPALIVE
XLKUPDLT	DSNTIPI	X LOCK FOR SEARCHED U/D

For most parameters, the change takes effect immediately. For the following parameters, the change is not immediate:

- PARTKEYU
- SYSADM and SYSADM1
- CACHEDYN
- MAXKEEPD
- XLKUPDLT

For more information about these system parameters, see *DB2 Installation Guide*.

Other availability, scalability, and performance enhancements

Version 8 of DB2 introduces the following additional enhancements:

- When trigger processing occurs for conditional triggers, performance is improved because the processing requires fewer work files than in previous versions.
- New messages help you monitor long-running units of recovery during backout processing.
- The ability to lock partitioned table spaces at the partition level improves data availability.
- The RECOVER utility can restore concurrent copies much faster when you specify the new CURRENTCOPYONLY option.
- Several data availability enhancements have been added to the CHECK INDEX utility in the form of SHRLEVEL CHANGE, DRAIN_WAIT, RETRY, and RETRY_DELAY options.

Chapter 2. Easier development and integration of e-business applications

Version 8 of DB2 UDB for z/OS facilitates easier development and integration of your e-business applications through various functional enhancements. The following topics provide additional information:

Changes to SQL limits

Many SQL limits are greatly increased in Version 8. Increases in some of these limits improve availability, scalability, and performance. Increases in other limits improve flexibility, productivity, portability, and DB2 UDB family consistency, as highlighted in Table 10.

Table 10. Changes to SQL limits

Entity	Previous limit	New limit
Maximum length of an SQL identifier	18 bytes	128 bytes
Maximum length of a character string constant	255 bytes	32 704 UTF-8 bytes
Maximum length of a hexadecimal character constant	254 hexadecimal digits	32 704 hexadecimal digits
Maximum length of a graphic string constant	124 bytes	32 704 UTF-8 bytes
Maximum length of a table name	18 bytes	128 bytes
Maximum length of a column name	18 bytes	30 bytes
Maximum length of an alias or view name	18 bytes	128 bytes
Maximum length of an index key	255 bytes	2000 bytes
Maximum length of an SQL statement	32 KB	2 MB
Maximum length of a predicate	255 bytes	32 704 bytes
Maximum number of tables in join	15	225
Maximum length of a condition name	18 bytes	128 bytes
Maximum length of a host identifier	64 bytes	128 bytes
Maximum length of an SQL label	18 bytes	128 bytes
Maximum length of an SQL parameter name	18 bytes	128 bytes
Maximum length of an SQL variable name	18 bytes	128 bytes
Maximum length of CURRENT PACKAGESET special register	18 bytes	128 bytes
Maximum length of CURRENT PATH special register	254 bytes	2048 bytes

When SQL statements that are larger than 32 KB are passed to DB2 on PREPARE
and EXECUTE IMMEDIATE statements, they are passed in CLOBs and DBCLOBs.
For more information about handling large SQL statements, see the EXECUTE
IMMEDIATE and PREPARE statements in *DB2 SQL Reference*.

SQL enhancements

DB2 Version 8 introduces the following enhancements to SQL:
- "SELECT from INSERT statement"
- "Sequence objects" on page 34
- "Identity column enhancements" on page 36
- "DISTINCT predicate" on page 36
- "Support for scalar fullselect" on page 37
- "Multiple-row INSERT and FETCH statements" on page 39
- "Common table expressions" on page 43
- "GET DIAGNOSTICS statement" on page 44
- "Dynamic scrollable cursors" on page 46
- "SQL procedural language enhancements" on page 46
- "More frequent use of indexes" on page 48
- "Longer and more complex SQL statements" on page 49
- "Multiple DISTINCT keywords" on page 49
- "Expressions in the GROUP BY clause" on page 49
- "Fewer restrictions for column functions (aggregate functions)" on page 49
- "Qualified column names in the INSERT statement" on page 50
- "ORDER BY clause for the SELECT INTO statement" on page 50
- "Additional input format for timestamp strings" on page 50
- "Explicitly defined ROWID columns no longer required for LOBs" on page 51
- "Comments for plans and packages" on page 51
- "Implicit dropping of declared global temporary tables at commit" on page 51
- "SQL changes for multilevel security with row-level granularity" on page 52
- "Comments in SQL statements" on page 52
- "Encrypting and decrypting data" on page 53
- "Greater control over locking for queries" on page 53

For a complete summary of the changes to SQL in Version 8, see Appendix C, "Changes to SQL," on page 103.

The new enhancements to DB2 not only provide significant new function but also increase SQL consistency across the DB2 UDB family of relational database products. If you are writing portable applications, see *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*. This book describes the SQL that is common to the DB2 UDB family of products, including rules and limits for preparing portable applications.

SELECT from INSERT statement

You can select values from rows that are being inserted by specifying 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. When you insert one or more new rows into a table, you can retrieve the following values from the result table:
- Any column values that are the result of an expression
- 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

- The value of an automatically generated column, such as a ROWID or identity column

Example: Assume that an EMPLOYEE table is defined with the following statement:

```
CREATE TABLE EMPLOYEE
  (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);
```

Assume that you need to insert a row for a new employee into the EMPLOYEE table. To determine the values for the generated EMPNO, HIRETYPE, and HIREDATE columns, use the following statement, which demonstrates use of the INSERT statement within the SELECT statement:

```
SELECT EMPNO, HIRETYPE, HIREDATE
  FROM FINAL TABLE (INSERT INTO EMPLOYEE (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.

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 INSERT statement that is within the SELECT 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: You can retrieve all the values for a row that is inserted into a structure by using the following statement:

```
EXEC SQL SELECT * INTO :empstruct
  FROM FINAL TABLE (INSERT INTO EMPLOYEE (NAME, SALARY, DEPTNO, LEVEL)
                    VALUES('Mary Smith', 35000.00, 11, 'Associate'));
```

For this example, :empstruct is a host variable structure that is declared with variables for each of the columns in the EMPLOYEE table.

Selecting values when you insert multiple rows

If you are writing an application program and want to retrieve values from the insertion of multiple rows, you need to declare a cursor so that the INSERT statement is in the FROM clause of the SELECT statement of the cursor.

Example: To see the values of the ROWID columns that are inserted into the employee photo and resume table, you can declare a cursor by using the following statement:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT EMP_ROWID
  FROM FINAL TABLE (INSERT INTO DSN8810.EMP_PHOTO_RESUME (EMPNO)
                    SELECT EMPNO FROM DSN8810.EMP);
```

Primary keys and foreign keys

By using the INSERT statement within the SELECT statement, you can insert a row into a parent table with its primary key defined as a DB2-generated identity

column, and you can retrieve the value of the primary or parent key. You can then use this generated value as a foreign key in a dependent table.

Example: 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 an INSERT statement within the following SELECT 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;
```

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. The sequence of numeric values is generated in a monotonically ascending or descending order. Sequences, unlike identity columns, are not associated with tables. Applications refer to a sequence object to get its current value or the next value. The relationship between sequences and tables is controlled by the application, not by DB2.

Creating a sequence object

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 you 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 parameter determines the first value that DB2 generates. The values advance by the INCREMENT BY parameter in ascending or descending order.

The MINVALUE and MAXVALUE parameters define the minimum and maximum values that DB2 generates. The CYCLE or NO CYCLE parameters define whether DB2 wraps values when it generates the incremented 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.

Referencing a sequence object

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 in an SQL statement generates and returns the next value for the specified sequence. If an SQL statement contains multiple instances of a NEXT VALUE expression with the same sequence name, the sequence value increments only once for that statement.
- A PREVIOUS VALUE expression in an SQL statement returns the most recently generated value for the specified sequence from a prior NEXT VALUE expression (for that sequence) in a previous SQL statement within the current application process.

You can specify a NEXT VALUE or PREVIOUS VALUE expression in a SELECT clause, within a VALUES clause of an INSERT statement, within the SET clause of an UPDATE statement (with certain restrictions), or within a SET host-variable statement.

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 generate 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. The PREVIOUS VALUE expression in the second INSERT statement retrieves that same value because it was the sequence number that was most recently generated.

Identity column enhancements

With Version 8, the identity column has some new attributes which can be set with the CREATE TABLE statement. See Table 23 on page 104 for details about the new attributes that you can set with the CREATE TABLE statement. At some point, you might need to change the attributes of an identity column. With Version 8, you can use the ALTER TABLE statement with the ALTER COLUMN clause to change all of the attributes of an identity column except the data type, as follows:

- Restart the column values from the new value
- Change whether values for the column are always generated by DB2 or are generated only by default
- Change the number by which the column value increments
- Change the minimum value, or change to no minimum value
- Change the maximum value, or change to no maximum value
- Change to allow the minimum value to be less than or equal to the maximum value
- Set the column value to cycle, or change to no cycling
- Change the CACHE value (the number of column values for DB2 to preallocate in memory), or change to NO CACHE (no preallocation)
- Specify that the column values are generated in order of request, or specify that the column values do not need to be generated in order of request

Changing the data type of an identity column requires that you drop and then re-create the table. For more information see “Schema evolution” on page 2.

DISTINCT predicate

You can use the DISTINCT predicate to compare null values. Two forms of the DISTINCT predicate are:

IS DISTINCT FROM
Creates an expression where both values are not equal or one value is null.

IS NOT DISTINCT FROM
Creates an expression where one value is equal to another value or both values are null. This predicate can also be written as **NOT(***value* **IS DISTINCT FROM** *value***)**

The DISTINCT predicate simplifies the SQL that you need to write when you need to find values that might be null. Because one null value is not considered equal to another null value, you cannot directly compare two null values using the = predicate. You also cannot test for a null value by using a host variable with an indicator variable that is set to -1.

Example: The following code selects the phone numbers of all employees except those who do not have a phone number:

```
MOVE -1 TO PHONE-IND.
EXEC SQL
  SELECT LASTNAME
    INTO :PGM-LASTNAME
    FROM DSN8810.EMP
    WHERE PHONENO = :PHONE-HV:PHONE-IND
END-EXEC.
```

You can use the IS NULL predicate to select employees who have no phone number, as in the following statement:

```
EXEC SQL
  SELECT LASTNAME
    INTO :PGM-LASTNAME
    FROM DSN8810.EMP
    WHERE PHONENO IS NULL
END-EXEC.
```

This works well if you are only trying to find values that are null. However, if you need to find values that are equal to a specific value and values that are null, the SQL statement that you must write becomes much more complex.

Example: To select employees whose phone numbers are equal to the value of :PHONE-HV and employees who have no phone number (as in the preceding example), you would need to code two predicates, one to handle the non-null values and another to handle the null values, as in the following statement:

```
EXEC SQL
  SELECT LASTNAME
  INTO :PGM-LASTNAME
  FROM DSN8810.EMP
  WHERE (PHONENO = :PHONE-HV AND PHONENO IS NOT NULL AND :PHONE-HV IS NOT NULL)
         OR
        (PHONENO IS NULL AND :PHONE-HV:PHONE-IND IS NULL)
END-EXEC.
```

You can use the DISTINCT predicate to get the same results. The following statement uses the NOT form of the IS DISTINCT FROM predicate to simplify the preceding example:

```
EXEC SQL
  SELECT LASTNAME
    INTO :PGM-LASTNAME
    FROM DSN8810.EMP
    WHERE PHONENO IS NOT DISTINCT FROM :PHONE-HV:PHONE-IND
END-EXEC.
```

Support for scalar fullselect

A *scalar fullselect* is a fullselect, enclosed in parentheses, that returns a single row consisting of a single column. You can now use a scalar fullselect wherever expressions are allowed, with some limitations. If the scalar fullselect does not return a row, the result is the null value. At run time, if the scalar fullselect returns more than one row, DB2 issues an error.

The following four tables, PARTS, PRODUCTS, PARTPRICE, and INVENTORY, are used in the examples in this section:

- **PARTS**

```
PART     PROD#  SUPPLIER
======== ====== =================
WIRE     10     ACWF
```

```
OIL      160    WESTERN_CHEM
MAGNETS  10     BATEMAN
PLASTIC  30     PLASTIC_CORP
BLADES   205    ACE_STEEL
```

- **PRODUCTS**

```
PROD#  PRODUCT            PRICE
====== ================== =====
505    SCREWDRIVER         3.70
30     RELAY               7.55
205    SAW                18.90
10     GENERATOR          45.75
```

- **PARTPRICE**

```
PART      PROD#  SUPPLIER           PRICE
========= ====== ================== =====
WIRE      10     ACWF                3.50
OIL       160    WESTERN_CHEM        1.50
MAGNETS   10     BATEMAN            59.50
PLASTIC   30     PLASTIC_CORP        2.00
BLADES    205    ACE_STEEL           8.90
```

- **INVENTORY**

```
PART      PROD#  SUPPLIER          ONHAND#
========= ====== ================= =======
WIRE      10     ACWF                    8
OIL       160    WESTERN_CHEM           25
MAGNETS   10     BATEMAN                 3
PLASTIC   30     PLASTIC_CORP            5
BLADES    205    ACE_STEEL              10
```

Example: Scalar fullselects in a WHERE clause: Find which products have prices in the range of at least twice the lowest price of all the products and at most half the price of all the products.

```
SELECT PRODUCT, PRICE
  FROM PRODUCTS A
  WHERE
    PRICE BETWEEN 2 * (SELECT MIN(PRICE) FROM PRODUCTS)
          AND 0.5 * (SELECT MAX(PRICE) FROM PRODUCTS);
```

The result is:

```
PRODUCT            PRICE
================== =====
RELAY               7.55
SAW                18.90
```

Example: Scalar fullselect in a SELECT list: For each part, find its price and its inventory.

```
SELECT PART,
  (SELECT PRICE   FROM PARTPRICE WHERE PART = A.PART),
  (SELECT ONHAND# FROM INVENTORY WHERE PART = A.PART)
  FROM PARTS A;
```

The result is:

```
PART           PRICE  ONHAND#
============== =====  =======
WIRE            3.50        8
OIL             1.50       25
MAGNETS        59.50        3
PLASTIC         2.00        5
BLADES          8.90       10
```

Example: Scalar fullselect in the SET clause of an UPDATE statement: Give a 20% discount to the parts that have a large inventory (greater than 20), and raise the price by 10% on the parts that have a small inventory (less than 7).

```
CREATE TABLE NEW_PARTPRICE LIKE PARTPRICE;

INSERT INTO NEW_PARTPRICE SELECT * FROM PARTPRICE;

UPDATE NEW_PARTPRICE N
  SET PRICE =
    CASE
      WHEN((SELECT ONHAND# FROM INVENTORY WHERE PART=N.PART) < 7)
        THEN 1.1 * PRICE
      WHEN((SELECT ONHAND# FROM INVENTORY WHERE PART=N.PART) > 20)
        THEN .8 * PRICE
      ELSE PRICE
    END;

SELECT * FROM NEW_PARTPRICE;
```

The result is:

```
PART      PROD#  SUPPLIER           PRICE
========= ====== ================== =====
WIRE      10     ACWF                3.50
OIL       160    WESTERN_CHEM        1.20
MAGNETS   10     BATEMAN            65.45
PLASTIC   30     PLASTIC_CORP        2.20
BLADES    205    ACE_STEEL           8.90
```

Restrictions: You cannot use scalar fullselects in the following cases:

- In an expression that is an argument of an aggregate function
- In the join-condition expression of an ON clause
- In the grouping expression in a GROUP BY clause
- In the sort-key expression of an ORDER BY clause
- In the RETURN statement of a CREATE FUNCTION statement
- In a CHECK condition in CREATE TABLE and ALTER TABLE statements
- In a CREATE VIEW statement that includes the WITH CHECK OPTION

Multiple-row INSERT and FETCH statements

You can enhance the performance of your application programs by using multiple-row INSERT and FETCH statements to request that DB2 send multiple rows of data at one time to and from the database. For local applications, using these multiple-row statements results in fewer accesses of the database. For distributed applications, using these multiple-row statements results in fewer network operations and a significant improvement in performance. This section provides an overview of how you can:

- Insert multiple rows of data from host variable arrays that have been declared and populated in your application program into the database; see "Inserting multiple rows" on page 40.
- Fetch multiple rows of data from the database into host variable arrays that have been declared or dynamically allocated in your program; see "Fetching multiple rows" on page 40.

To use a host variable array in an SQL statement, specify a host variable array that is declared according to host language rules. You can specify host variable arrays in C, C++, COBOL, and PL/I application programs. You must declare the array in the host program before you use it in an SQL statement.

You can also use a storage area that you allocate dynamically when you use a descriptor to describe the data areas that you want DB2 to use to insert or place the data. You can specify a descriptor in assembler, C, C++, COBOL, and PL/I application programs. You must include an SQL descriptor area (SQLDA) in the host program.

Inserting multiple rows

You can use a form of the INSERT 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 DSN8810.ACT
    (ACTNO, ACTKWD, ACTDESC)
    VALUES (:HVA1, :HVA2, :HVA3 :IVA3)
    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.

Assume also that the indicator variable array IVA3 has been declared and populated to indicate whether null values are inserted into the ACTDESC column. Use indicator variable arrays with host variable arrays in the same way that you use indicator variables with host variables. An indicator variable array must have at least as many entries as its host variable array.

You can use the multiple-row INSERT statement both statically and dynamically. If you prepare and execute the INSERT statement, you can code the EXECUTE statement to use either host variable arrays or an SQL descriptor (SQLDA). If you use host variable arrays, each host variable array in the USING clause of the EXECUTE statement represents a parameter marker in the INSERT statement. If you use an SQLDA, the host variable in the USING clause of the EXECUTE statement names the SQLDA that describes the parameter markers in the INSERT statement.

Fetching multiple rows

You can retrieve multiple rows of data by using a row-set positioned cursor. A row-set positioned cursor retrieves zero, one, or more rows at a time, as a row set, from the result table of the cursor into host variable arrays. You can reference all of the rows in the row set, or only one row in the row set, when you use a positioned DELETE or positioned UPDATE statement after a FETCH statement that retrieves row sets.

A multiple-row FETCH statement can be used to copy a row set 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 to be retrieved

Declaring a row-set positioned cursor: You must first declare a row-set positioned cursor before you can retrieve row sets of data. To enable a cursor to fetch row sets, use the WITH ROWSET POSITIONING clause in the DECLARE CURSOR statement.

Example: The following statement declares a row set cursor:

```
EXEC SQL
  DECLARE C1 CURSOR
    WITH ROWSET POSITIONING FOR
    SELECT EMPNO, LASTNAME, SALARY
      FROM DSN8810.EMP
END-EXEC.
```

To tell DB2 that you are ready to process the first row set of the result table, execute the OPEN statement in your program. DB2 then uses the SELECT statement within the DECLARE CURSOR statement to identify the rows in the result table.

Using a multiple-row FETCH statement with host variable arrays: When your program executes a FETCH statement with the ROWSET keyword, the cursor is positioned on a row set in the result table. That row set is called the *current row set*. Declare the dimension of each of the host variable arrays to be greater than or equal to the number of rows that are to be retrieved.

Example: The following FETCH statement 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.
```

Using a multiple-row FETCH statement with a descriptor: Suppose that you want to dynamically allocate the necessary storage for the arrays of column values that are to be retrieved from the employee table. You must do the following steps:

1. Declare an SQLDA structure.
2. Dynamically allocate the SQLDA and the necessary arrays for the column values.
3. Set the fields in the SQLDA for the column values that are to be retrieved.
4. Open the cursor.
5. Fetch the rows.

After allocating the SQLDA and the necessary arrays for the column values, you must set the fields in the SQLDA.

Example: After the OPEN statement, the program fetches the next row set by using the following statement:

```
EXEC SQL
  FETCH NEXT ROWSET FROM C1
  FOR 20 ROWS
  USING DESCRIPTOR :outsqlda;
```

The USING clause of the FETCH statement names the SQLDA that describes the columns that are to be retrieved.

Using row-set positioned UPDATE statements: After your program executes a FETCH statement to establish the current row set, you can use a positioned UPDATE statement with either of the following clauses:

- WHERE CURRENT OF *cursor-name* to update:
  - a single row if the cursor is on a single row
  - all the rows of a row set if the cursor is on a row set
- WHERE CURRENT OF *cursor-name* FOR ROW *n* OF ROWSET to update only row *n* of the current row set

Updating all rows of the current row set: The following positioned UPDATE statement uses the WHERE CURRENT OF clause:

```
EXEC SQL
  UPDATE DSN8810.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 row set of the result table. If the cursor is positioned on a row, that row is updated. If the cursor is positioned on a row set, all of the rows in the row set are updated.

Updating a specific row of the current row set: The following positioned UPDATE statement uses the WHERE CURRENT OF *cursor* FOR ROW *n* OF ROWSET clause:

```
EXEC SQL
  UPDATE DSN8810.EMP
    SET SALARY = 50000
    WHERE CURRENT OF C1 FOR ROW 5 OF ROWSET
END-EXEC.
```

When the UPDATE statement is executed, the cursor must be positioned on a row set of the result table. The specified row (in the example, row 5) of the current row set is updated.

Using row-set positioned DELETE statements: After your program executes a FETCH statement to establish the current row set, you can use a positioned DELETE statement with either of the following clauses:

- WHERE CURRENT OF *cursor-name* to delete:
  - a single row if the cursor is on a single row
  - all the rows of a row set if the cursor is on a row set
- WHERE CURRENT OF *cursor-name* FOR ROW *n* OF ROWSET to delete only row *n* of the current row set

Deleting all rows of the current row set: The following positioned DELETE statement uses the WHERE CURRENT OF clause:

```
EXEC SQL
  DELETE FROM DSN8810.EMP
    WHERE CURRENT OF C1
END-EXEC.
```

When the DELETE statement is executed, the cursor must be positioned on a row or row set 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 row set, all of the rows in the row set are deleted, and the cursor is positioned before the next row set of its result table.

Deleting a single row of the current row set: The following positioned DELETE statement uses the WHERE CURRENT OF *cursor* FOR ROW *n* OF ROWSET clause:

```
EXEC SQL
  DELETE FROM DSN8810.EMP
    WHERE CURRENT OF C1 FOR ROW 5 OF ROWSET
END-EXEC.
```

When the DELETE statement is executed, the cursor must be positioned on a row set of the result table. The specified row of the current row set is deleted, and the cursor remains positioned on that row set. The deleted row (in the example, row 5 of the row set) cannot be retrieved or updated.

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 for the SELECT, INSERT, and CREATE VIEW statements.

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 a common table expression 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 desired 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

Using WITH instead of CREATE VIEW

Using the WITH clause to create a common table expression saves you the overhead of needing to create and drop 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.

Using a common table expression for a result table that is based on host variables or is shared in a fullselect

You can use a common table expression when you need to find information that is based on an intermediate result table that is derived from the values of host variables. This results in a decrease in the overhead that is associated with creating a temporary table to hold the intermediate results, and writing additional queries to derive your final result. This overhead is also saved when you need to share the same result table in a fullselect.

Using recursive SQL

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*. Queries that use recursion are useful in applications like bill-of-materials applications, network planning applications, and reservation systems.

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 types, lengths, and CCSIDs of the column names from the common table expression that are referenced in the iterative fullselect must match.
- The UNION statements must be UNION ALL.
- Outer joins must not be part of any recursion cycle.
- Subqueries must not be part of any recursion cycle.

Introducing an infinite loop might occur when you develop a recursive common table expression. A recursive common table expression is expected to include a predicate that will prevent an infinite loop. A warning is issued if one of the following objects 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*

GET DIAGNOSTICS statement

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 connection information about the SQL statement

The GET DIAGNOSTICS statement is also useful for getting information about long names, which do not fit in the SQLCA. In addition to information about long names, you can use the GET DIAGNOSTICS statement to obtain the following new information:

- An indication of the last row in a multi-row FETCH statement
- The number of parameter markers in a prepared statement
- The actual number of result sets that are returned by a stored procedure
- The number of rows in the result table
- The attributes of a cursor
- The number of errors and error information that is generated by the previous statement
- The message IDs and text that is generated by the previous 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.

Use the GET DIAGNOSTICS statement to handle multiple SQL errors that might result from the execution of a single SQL statement. This method is especially useful for diagnosing problems that result from a multiple-row INSERT that is specified as NOT ATOMIC CONTINUE ON SQLEXCEPTION.

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. First, you must declare target host variables with data types that are compatible with the data types of the requested items of diagnostic information. 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 Figure 4, the first GET DIAGNOSTICS statement returns the number of rows that are inserted and the number of conditions that are 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 DSN8810.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);
 }
```

Figure 4. GET DIAGNOSTICS statement for a multi-row INSERT statement

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 -253, 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 = -253, SQLSTATE = 22003, ROW NUMBER = 4
```

You can also retrieve the MESSAGE_TEXT item for this example, indicating that the value 32768 for the input variable is too large for the target column ACTNO.

Dynamic scrollable cursors

When you declare a cursor as SENSITIVE, you can declare it as either STATIC or DYNAMIC. The SENSITIVE DYNAMIC cursor follows the dynamic scrollable cursor model:

- 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 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.

Using a dynamic scrollable cursor, you can fetch newly inserted rows but you cannot fetch deleted rows. In contrast, with a static scrollable cursor, you cannot fetch newly inserted rows, and rows that have been deleted are indicated as holes in the result table of the cursor.

Example: The following statement shows a declaration for a sensitive dynamic scrollable cursor:

```
EXEC SQL DECLARE C2 SENSITIVE DYNAMIC SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8810.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declaring a cursor as SENSITIVE DYNAMIC has the following effects:

- Because the associated FETCH statement executes on 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.
- If you specify an ORDER BY clause for a SENSITIVE DYNAMIC cursor, DB2 might choose an index access path if the ORDER BY is fully satisfied by an existing index.

SQL procedural language enhancements

Version 8 of DB2 UDB for z/OS provides the following improvements to the SQL procedural language:

Extending the length of an SQL procedure statement

With earlier releases of DB2 UDB for z/OS, an SQL procedure needed to be completely specified in a CREATE PROCEDURE statement that was limited to 32 KB. In addition, because the procedure body of a CREATE PROCEDURE statement contains the source statements for the procedure, each of those statements was limited to 32 KB.

Version 8 of DB2 UDB for z/OS extends the length of any SQL statement to 2 MB, including a CREATE PROCEDURE statement. In addition, the length of individual SQL procedural statements, consisting of SQL control statements and SQL statements in the procedure body, is extended to 2 MB. If you specify an SQL control statement as the procedure body, you can include multiple SQL procedural statements within that control statement, each of which is now extended to 2 MB. This enhancement significantly increases the power and flexibility of SQL procedures.

Handling SQL conditions in an SQL procedure

You can handle SQL errors and warnings in an SQL procedure by using the following techniques:

- You can include a RETURN statement in an SQL procedure to return an integer status value to the caller; see "Using the RETURN statement for the procedure status."
- You can include a SIGNAL or RESIGNAL statement to raise a specific SQLSTATE and to define the message text for that SQLSTATE; see "Using SIGNAL or RESIGNAL to raise a condition."
- You can include handlers to tell the procedure to perform some other action when an error occurs. This section describes how you can use the GET DIAGNOSTICS statement in a handler; see "Using GET DIAGNOSTICS in a handler" on page 48. You can use the GET DIAGNOSTICS statement in the body of an SQL procedure and generally anywhere in an application program. For more information about using the GET DIAGNOSTICS statement, see "GET DIAGNOSTICS statement" on page 44.

Using the RETURN statement for the procedure status: You can use the RETURN statement in an SQL procedure to return an integer status value. If you include a RETURN statement, DB2 sets the SQLCODE in the SQLCA to 0, and the caller must retrieve the return status of the procedure in either of the following ways:

- By using the RETURN_STATUS item of GET DIAGNOSTICS to retrieve the return value of the RETURN statement
- By retrieving the first SQLERRD field in the SQLCA, which contains the return value of the RETURN statement

If you do not include a RETURN statement in an SQL procedure, by default, DB2 sets the return status to 0 for an SQLCODE that is greater than or equal to 0, and to -1 for an SQLCODE that is less than 0.

Using SIGNAL or RESIGNAL to raise a condition: 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.
- 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.

Recommendation: You can use any valid SQLSTATE value in a SIGNAL or RESIGNAL statement; however, using the range of SQLSTATE values that are reserved for application programs is recommended.

Example: 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, as in the following example:

```
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
```

In this example, the overflow condition is declared for SQLSTATE 22003, and the handler is declared for the overflow condition. The RESIGNAL statement in the handler sets the new SQLSTATE value for overflow to 22375.

Using GET DIAGNOSTICS in a handler: 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. For information about using the GET DIAGNOSTICS statement anywhere in a DB2 application program, see "GET DIAGNOSTICS statement" on page 44.

Example: Suppose that you create an SQL procedure named divide1 that computes the result of the division of two integers. You include the following GET DIAGNOSTICS statement 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(70))
  LANGUAGE SQL
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      GET DIAGNOSTICS CONDITION 1 :divide_error = MESSAGE_TEXT;
    SET divide_result = numerator / denominator;
  END
```

Debugging an SQL procedure

You can remotely debug SQL stored procedures that execute on DB2 UDB for z/OS servers using the SQL Debugger. The SQL Debugger is integrated into various client development platforms including DB2 Development Center. With the SQL Debugger, you can observe the execution of SQL procedure code, set break points for lines and view or modify variable values.

For up-to-date information on the SQL Debugger refer to ″DB2 Development Center″ topics at the following Web page: http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

More frequent use of indexes

Version 8 of DB2 introduces the concept of *volatile tables.* Volatile tables are defined with the keyword VOLATILE and contain clusters of rows that logically belong

together. Within these clusters, the rows are intended to be accessed in the same order every time. For these tables, DB2 uses index access whenever possible, regardless of the impact on performance.

Longer and more complex SQL statements

In addition to other limit-breaking support in Version 8, you can now have SQL statements that are up to 2 MB in length. A number of the Version 8 capabilities required increasing the previous limit on the size of an SQL statement, which was 32 KB. For example, support for long names and 4096 partitions requires much more space. Other changes in DB2 allow much larger structures and thus much larger statements. SQL statements that are too large or too complex should now be very unusual.

Multiple DISTINCT keywords

In previous releases of DB2, you could specify only one DISTINCT keyword on the SELECT clause or the HAVING clause of a query. In Version 8, you can specify more than one DISTINCT keyword for a given query. This enhancement improves performance. The ability to specify multiple DISTINCT keywords eliminates the need to code multiple queries, especially when you need to retrieve distinct values for multiple columns to which you want to apply aggregate functions such as AVG, COUNT, and SUM.

Example: Using the sample employee table, suppose that you want to determine the average number of employees per department and the number of different jobs that these employees hold. Instead of using two queries, you could use the following subselect to find that information:

```
SELECT COUNT(EMPNO)/COUNT(DISTINCT(WORKDEPT)), COUNT(DISTINCT(JOB))
  FROM DSN8810.EMP;
```

Expressions in the GROUP BY clause

In previous releases of DB2, you could specify only columns in the GROUP BY clause of a query. In Version 8, you can use an expression in the GROUP BY clause to specify how the rows are to be grouped. The ability to use an expression makes coding your applications easier because you no longer need to use a nested table expression or view to provide a result table with the expression as a column of the result and then specify the column in the GROUP BY clause. In addition, the expressions in the GROUP BY can be referenced in the SELECT, HAVING, and ORDER BY clauses if the reference specifies only one value for each group.

Example: Using the sample employee table, suppose that you want to find the average salary for all employees that were hired in the same year. You could use the following subselect to group the rows by the year of hire:

```
SELECT AVG(SALARY), YEAR(HIREDATE)
  FROM DSN8810.EMP
  GROUP BY SUBSTR(VARCHAR(HIREDATE),1,4);
```

Fewer restrictions for column functions (aggregate functions)

The argument of a column function is a set of like values that is derived from an expression. Previous to Version 8 of DB2, the expression for the argument was required to include a reference to a column (hence, the term *column function*). In Version 8, you no longer need to specify a column name in the expression. Because a column reference is no longer required, column functions are now being called *aggregate functions*.

Example: Assume that a table exists that contains one column (C1) that is defined as an integer and that all the values in C1 are 5. Invoking an aggregate function with C1 as the argument is similar to invoking the same function with a constant value of 5. For example, assuming that the table has 10 rows, the result for both of the following functions should be 50:

```
SUM(C1)
SUM(5)
```

Qualified column names in the INSERT statement

In previous releases of DB2, you could not qualify the name of columns when inserting data into a column. In Version 8, you can use qualified column names in an INSERT statement just like you can in an UPDATE statement.

Example: Assume that MYTABLE.YEMP is a copy of the sample employee table. You want to insert a new row into the table. You might use the following INSERT statement to add information for a new employee:

```
INSERT INTO MYTABLE.EMP (YEMP.EMPNO, YEMP.FIRSTNME, MIDINIT, MYTABLE.YEMP.LASTNAME)
  VALUES ('200540', 'SUSAN', 'S', 'WALKER');
```

ORDER BY clause for the SELECT INTO statement

The SELECT INTO statement must produce a result that contains a single row. Previous to Version 8 of DB2 UDB, you could specify the FETCH FIRST 1 ROW clause to ensure that only a single row was returned if the result set of the query could result in more than one row. However, you could not specify the ORDER BY clause to affect which row was returned. With Version 8, you can now specify ORDER BY. When you use both the FETCH FIRST 1 ROW and ORDER BY clauses, the result set is ordered first and then the first row is returned.

Example: Using the sample employee table, for all employees with a salary of more than $40000, put the salary of the employee who has been employed the longest in host variable HV1:

```
SELECT SALARY
  FROM DSN8810.EMP
  INTO :HV1
  WHERE SALARY > 40000
  ORDER BY HIREDATE
  FETCH FIRST ROW ONLY;
```

Additional input format for timestamp strings

In addition to using a dash to separate the date portion and the time portion of a timestamp string, you can also use a blank as the separator in Version 8 of DB2. In this alternate format, a colon is used to separate the hours from the minutes and the minutes from the seconds Therefore, DB2 accepts either of the following strings as a valid input representation of a timestamp value:

```
'2003-03-02-08.30.00.000000' or '20031-03-02 08:30:00.000000'
```

The ODBC and JDBC string representation of a timestamp uses the format in which the blank is the separator.

Example: Assume that you have a table named SALES that has a TRANSDATE column with a TIMESTAMP data type. You want to find all the transactions that were made before the timestamp value ’2003-01-01 00:00:00.000000’.

```
SELECT TRANSID
FROM SALES
WHERE TRANSDATE < '2003-01-01 00:00:00.000000';
```

Explicitly defined ROWID columns no longer required for LOBs

In Version 8 of DB2, you no longer need to explicitly define a ROWID column when you define a LOB column. If a ROWID column does not exist when you define a LOB column with either the ALTER TABLE or CREATE TABLE statement, DB2 implicitly generates a ROWID column. When DB2 generates the ROWID column, it is called a *hidden ROWID column*, and DB2:

- Creates the column with a name of DB2_GENERATED_ROWID_FOR_LOBS*nn*.

  DB2 appends *nn* only if the column name already exists in the table, replacing *nn* with 00 and incrementing by 1 until the name is unique within the row.
- Defines the column as GENERATED ALWAYS.
- Appends the column to the end of the row after all the other explicitly defined columns.

If you add a ROWID column to a table that already has a hidden ROWID column, DB2 ensures that the values in the two ROWID columns are identical. If the table has a hidden ROWID column and the ROWID column that you add is defined as GENERATED BY DEFAULT, DB2 changes the hidden ROWID column to have the GENERATED BY DEFAULT attribute.

A hidden ROWID column is not visible in SQL statements unless you refer to the column directly by name. For example, assume that DB2 generated a hidden ROWID column named DB2_GENERATED_ROWID_FOR_LOBS for table MYTABLE. The result table for a SELECT * statement for table MYTABLE would not contain that ROWID column. However, the result table for SELECT COL1, DB2_GENERATED_ROWID_FOR_LOBS would include the hidden ROWID column.

Comments for plans and packages

In Version 8 of DB2, you can provide comments for plans and packages in the DB2 catalog. Support for comments for plans and packages simplifies documenting and tracking your objects. It also increases compatibility within the DB2 UDB family.

Example: Provide a comment for package MY_PACKAGE, which is in collection COLLIDA.

```
COMMENT ON PACKAGE COLLIDA.MY_PACKAGE IS 'This is my package';
```

When adding comments for packages, you must qualify the package name with the collection ID.

Implicit dropping of declared global temporary tables at commit

In Version 8 of DB2, you can specify that DB2 is to implicitly drop declared global temporary tables at a commit operation. When you specify the new ON COMMIT DROP TABLE clause of the DECLARE GLOBAL TEMPORARY TABLE statement, DB2 drops the declared global temporary table at commit if no open cursors on the table are defined as WITH HOLD.

This enhancement is particularly important for distributed applications and stored procedures because clean up can occur when cursors are closed. For example, consider a self-contained stored procedure that declares several temporary tables and cursors for the result sets that are defined on those temporary tables. An invoker of the stored procedure can access the returned result sets and then issue a

COMMIT statement that would result in DB2 automatically dropping the declared global temporary tables, assuming that they are declared with ON COMMIT DROP TABLE. Thus, the invoker of the stored procedure, who did not define the declared global temporary tables, does not need to know the names of the declared global temporary tables to explicitly drop them.

Example: Define a declared temporary table with column definitions for an employee number, salary, commission, and bonus. Indicate that the temporary table is to be implicitly dropped at a commit operation if no open cursors on the table are defined as WITH HOLD.

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMP_EMP
  (EMPNO     CHAR(6) NOT NULL,
   SALARY    DECIMAL(9, 2),
   COMM      DECIMAL(9, 2),
   BONUS     DECIMAL(9, 2))
   CCSID EBCDIC
   ON COMMIT DROP TABLE;
```

SQL changes for multilevel security with row-level granularity

DB2 Version 8 introduces multilevel security with row-level granularity, which is described in "Multilevel security with row-level granularity" on page 58. When a table has multilevel security with row-level granularity, one column in the table contains the security label for each row. When you execute a CREATE TABLE or ALTER TABLE statement, you define the column that contains the security label with the CHAR(8) data type and with the AS SECURITY LABEL and NOT NULL WITH DEFAULT attributes.

Example: To add a security label column to the sample employee table, you might execute this ALTER TABLE statement:

```
ALTER TABLE DSN8810.EMP
  ADD EMP_SECLABEL CHAR(8) AS SECURITY LABEL NOT NULL WITH DEFAULT;
```

Comments in SQL statements

DB2 Version 8 allows any SQL statement to include SQL comments. Two consecutive hyphens (--) indicate that the characters after the hyphens are part of a comment. SQL comments must conform to the following rules:

- The two hyphens must be on the same line.
- The two hyphens must not be separated by a space.
- Comments can be started wherever a space is valid (except within a delimiter token or between EXEC and SQL).
- Comments must begin and end on the same line.
- Within a statement that is embedded in a COBOL program, the two hyphens must be preceded by a blank unless they begin a line.

Example: The following SELECT statement illustrates the use of a comment:

```
SELECT SALARY
  FROM DSN8810.EMP
  INTO :HV1
  WHERE SALARY > 40000
  ORDER BY HIREDATE -- this finds the employee who has been employed the longest
  FETCH FIRST ROW ONLY;
```

Encrypting and decrypting data

In Version 8, you can use the new ENCRYPT_TDES function to encrypt data as you update tables, and the DECRYPT_BIT, DECRYPT_CHAR, and DECRYPT_DB functions to decrypt encrypted data as you retrieve it from a table.

Using the ENCRYPT_TDES function you can set a password string and a hint string (to help you remember the password).

Example 1: Use the ENCRYPT_TDES function to encrypt an employe's social security number with an encryption password of 'Pacific' and a password hint of 'Ocean':

```
INSERT INTO EMP(SSN)
 VALUES ENCRYPT_TDES ('289-46-8832', 'Pacific', 'Ocean');
```

You can retrieve the password hint by using the new GETHINT function as follows:

```
SELECT GETHINT (SSN)
  FROM EMP;
```

This returns the value 'Ocean'.

To retrieve encrypted data from a table, you need to use the DECRYPT_BIT, DECRYPT_CHAR, or DECRYPT_DB function using the password that you specified in the ENCRYPT_TDES function.

Example 2: Use the DECRYPT_CHAR function to return the decrypted employe's social security number where the encryption password is 'Pacific':

```
SELECT DECRYPT_CHAR(SSN, 'Pacific')
 FROM EMP;
```

This returns the value '289–46–8832', which was encrypted in Example 1.

Greater control over locking for queries

In Version 8, you can use the USE AND KEEP EXCLUSIVE LOCKS, USE AND KEEP UPDATE LOCKS, or USE AND KEEP SHARE LOCKS clauses in a SELECT or SELECT INTO statement. These options are only valid when you use WITH RR or WITH RS. By using one of these clauses, you tell DB2 to acquire and hold an X, U, or S lock, respectively, on all of the qualified pages or rows.

With read stability (RS) isolation, a row or page that is rejected during stage 2 processing might still have a lock held on it, even though it has not returned to the application.

With repeatable read (RR) isolation, DB2 acquires locks on all pages or rows that are within the range of the selection expression.

All locks are help until the application commits. Although these options can reduce concurrency, these option can prevent some types of deadlocks and can better serialize access to data.

Unicode enhancements

In Version 8, DB2 UDB for z/OS offers uniform data management across geographic regions, greater compatibility between encoding schemes, and an ODBC driver with full Unicode support. The Unicode enhancements that provide these additional functions include support for Unicode parsing, support for multiple encoding schemes in a single SQL statement, and ODBC support for native Unicode.

Support for Unicode parsing

In DB2 Version 8, a Unicode parser replaces the EBCDIC parser that was used in Version 7. The new Unicode parsing scheme alleviates problems with the variant code set. The *variant code set* is the set of code points that are not represented by the same hexadecimal value on each EBCDIC code page. This code set includes special characters such as $, @, #, ¬, |, [,], {, and }.

Because a Unicode parser replaces the EBCDIC parser, Version 8 converts all SQL statements that are not currently encoded as Unicode UTF-8 to that format before parsing. The DB2 precompiler and coprocessor convert the coded character set identifier (CCSID) of source programs to CCSID 1208 (the CCSID for UTF-8 data). For this conversion, you specify the CCSID of the source program (a number from 1 to 65533 or 65535) in the new precompiler and coprocessor option CCSID. The default value for this CCSID option is the system EBCDIC CCSID that was specified in the DSNTIPF installation panel. If you connect to a server that does not support Unicode, before DB2 sends data it converts character strings to the EBCDIC system CCSID set of the non-Unicode server.

When you migrate DB2 from Version 7 to Version 8, you make the transition in three DB2 migration modes: compatibility mode, enabling-new-function mode, and new-function mode. These modes change the encoding scheme, compatibility, and functionality of your subsystem in the following ways:

Compatibility mode
All catalog and directory table spaces are encoded in EBCDIC, allowing the Version 8 subsystem to coexist with Version 7 subsystems. This mode does not enable new Version 8 functions.

Enabling-new-function mode
Catalog and directory table spaces are in either EBCDIC or Unicode UTF-8. Some table spaces remain in EBCDIC because they have not yet been converted to Unicode. This mode is the transitional period between encoding schemes. A DB2 subsystem that is in this mode cannot coexist with nor fall back to Version 7. This mode supports only a limited set of Version 8 functions to support the enabling process.

New-function mode
The following directory table spaces remain in EBCDIC in new-function mode:
- SYSIBM.SYSCOPY
- SYSIBM.SYSEBCDIC

All other catalog and directory table spaces are encoded in Unicode. A DB2 subsystem that is in this mode cannot coexist with nor fall back to Version 7. Additionally, a DB2 subsystem that is in this mode cannot coexist with nor return to Version 8 compatibility mode. New-function mode enables all new Version 8 functions. To disable Version 8 functionality, you can toggle

between new-function mode and enabling-new-function mode, but you cannot backout the processing that originally occurred in enabling-new-function mode.

The precompiler runs outside of DB2, so it cannot directly determine the current DB2 migration mode. To specify whether you want the precompiler to enable Version 8 syntax, you set an additional precompiler option NEWFUN to NO or YES. If you are migrating to Version 8, NO is the default value for this option. The last action of enabling-new-function mode is to rebuild the DSNHDECP module to specify YES for the NEWFUN default.

NO Setting NEWFUN to NO tells the precompiler to disable Version 8 functions. The resulting DBRM uses EBCDIC for SQL statements and is not marked Version 8 dependent under this option. DB2 Version 7 can bind this DBRM. NO is the default value for NEWFUN in compatibility mode and in enabling-new-function mode.

YES Setting NEWFUN to YES tells the precompiler to enable Version 8 functions. The resulting DBRM uses Unicode for SQL statements and is marked Version 8 dependent under this option. This DBRM is Version 8 dependent even if you are using no new Version 8 SQL. **The resulting DBRM can only be bound on a DB2 Version 8 server.** YES is the default value for NEWFUN in new-function mode.

Support for multiple CCSID sets in a single SQL statement

With Version 8, you can reference table objects from different encoding schemes in a single SQL statement. Table objects include tables, views, global temporary tables, declared temporary tables, materialized query tables, and user-defined table functions. DB2 supports EBCDIC, ASCII, and Unicode encoding schemes. A set of coded character set identifiers (CCSIDs) defines each encoding scheme. You can begin to reference table objects from different CCSID sets when DB2 is in enabling-new-function mode. Referencing these table objects requires no additional SQL syntax, but the rules for using multiple CCSIDs might change the semantics of certain SQL statements. Multiple CCSID sets can semantically affect the following SQL statements:

- ALTER TABLE ADD *materialized-query-definition*
- ALTER TABLE *materialized-query-alteration*
- CREATE GLOBAL TEMPORARY TABLE LIKE *view-table*
- CREATE TABLE *materialized-query definition*
- CREATE TABLE LIKE *view-table*
- CREATE VIEW
- DECLARE GLOBAL TEMPORARY TABLE AS *(fullselect)* DEFINITION ONLY
- DECLARE GLOBAL TEMPORARY TABLE LIKE *view-table*
- DELETE
- INSERT
- SELECT
- SELECT INTO
- UPDATE
- Scalar fullselect expressions

A CCSID set identifies an encoding scheme. When an SQL statement references multiple CCSID sets for comparison or to generate a result set, DB2 must choose a single CCSID set to represent the data. Because you do not explicitly define this CCSID set with additional SQL syntax, semantic rules define the CCSID set that DB2 is to use to represent or compare data.

A CCSID set consists of three different parts:

SBCS A single-byte character set in which each character is represented by a single byte.

Mixed A mixed-byte character set in which characters are represented by a combination of single and multiple bytes.

Graphic This part of the CCSID can represent one of the following data types:

- A double-byte character set (DBCS) in which each character is represented by a pair of bytes.
- A Unicode character set in which each character is represented by two or more bytes. Unicode graphic strings always use UTF-16 data which uses a CCSID of 1200.

String constants, special registers, and host variables for which no CCSID is specified in the SQLDA are associated with the application encoding scheme. Expressions that are not explicitly associated with a CCSID, such as SUBSTR and VARCHAR, produce results that use the same CCSID as the input string.

If you compare or combine data from multiple CCSIDs in an SQL statement, DB2 chooses the CCSID to which to convert data in the following process:

1. DB2 determines an appropriate CCSID set (SBCS, mixed, or graphic).
2. DB2 chooses a specific CCSID from that CCSID set.

DB2 first determines a CCSID set. Generally, DB2 chooses the graphic Unicode CCSID set when you compare or combine data from different CCSID sets. This rule does not apply, however, when an SQL statement references both column-based data and application data (such as string constants and special registers) in the same operation. In this case, DB2 converts the application data to the CCSID set of column-based data (which might or might not be Unicode).

DB2 then chooses the specific CCSID to which to convert data. If the SQL operation references column-based data and application data, DB2 always converts application data to the CCSID of the column-based data. If the SQL operation references only column-based data or only application data, DB2 performs **one** of the following conversions:

- For operations that reference graphic data, DB2 converts all string data to the graphic CCSID of the CCSID set that DB2 selected.
- For operations that reference only SBCS and mixed data, DB2 converts all data to the mixed data CCSID of the CCSID set that DB2 selected. However, if the MIXED DATA field in the DSNTIPF installation panel specifies NO (which is the default value), DB2 uses the following conversions for each CCSID set:
  - ASCII and EBCDIC mixed data operands are converted to SBCS. CCSID 65534 is used for both ASCII and EBCDIC mixed data subtypes in a MIXED DATA = NO environment as a placeholder. No conversion occurs to or from this CCSID.
  - Unicode SBCS CCSIDs are still converted to mixed data. The value that is specified for the MIXED DATA field in the DSNTIPF installation panel does not affect Unicode CCSIDs.
- For operations that reference SBCS data only, DB2 uses one of the following conversions:
  - If the CCSID set is Unicode, the SBCS operands are converted to UTF-8 mixed data.

– If the CCSID set is **not** Unicode, DB2 uses the SBCS CCSID.

Important: Any time DB2 converts a character set to another character set that contains fewer or different characters, you might lose data. If DB2 uses substitution characters, DB2 issues a warning. If DB2 cannot convert a character to the target CCSID, DB2 issues an error message (although in some special cases, only a warning is issued).

The CAST specification is extended in Version 8 to include the CCSID *integer* and CCSID *encoding-scheme* clauses. With these new clauses, you can specify the CCSID or CCSID set of the target data type in a CAST statement. The CCSID clauses in the CAST specification become available when DB2 enters enabling-new-function mode.

A new Unicode hexadecimal string constant supports graphic Unicode UTF-16 characters. You can specify UTF-16 hexadecimal constants in SQL statements using the form U*X'xxxx'*, where *xxxx* represents a group of four hexadecimal digits. You can use these digits in any multiple of four up 32704 digits. Each group of four digits is a UTF-16 character.

When you reference a UX string constant, a GX string constant, or use the CAST expression with the CCSID clause, DB2 treats your SQL statement as if it contains more than one CCSID. Data in this statement is converted using the two-step conversion process that is explained above.

To use Unicode hexadecimal string constants, you must run DB2 in new-function mode.

For more details about using multiple CCSID sets, see *DB2 SQL Reference* and *DB2 Installation Guide*.

DB2 ODBC support for native Unicode

The ODBC driver for Version 8 fully supports UTF-8 and UCS-2 Unicode encoding schemes. DB2 ODBC applications pass and store Unicode data directly without conversion. This support enables the following application programming features in DB2 ODBC:

- Update, insert, delete, and fetch operations on Unicode data through ODBC application variables
- Unicode strings within the ODBC application programming interface (which allow you to use Unicode SQL statements in your ODBC application)

The following DB2 ODBC elements support this new functionality:

- Suffix-W APIs to support UCS-2 data.
- New SQL_C_WCHAR data type to support UCS-2 data.
- A new initialization keyword, CURRENTAPPENSCH, which specifies the current encoding scheme (EBCDIC, ASCII, or Unicode). When you set this keyword to UNICODE, generic ODBC APIs support UTF-8 data.
- Additional SQLGetInfo() attributes to query the CCSID settings of the DB2 subsystem in each encoding scheme.

For additional information about DB2 ODBC Unicode support, see *DB2 ODBC Guide and Reference*.

Multilevel security with row-level granularity

In Version 8, DB2 UDB for z/OS introduces multilevel security with row-level granularity. Multilevel security allows you to classify objects and users with security labels that are based on hierarchical security levels and non-hierarchical security categories. Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and prevents users from declassifying information. Using multilevel security with row-level granularity, you can define security for DB2 objects and perform security checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

Requirement: You must have z/OS Version 1 Release 5 or later to use DB2 authorization with multilevel security with row-level granularity.

Advantages of multilevel security

Multilevel security with row-level granularity offers the following advantages:

- Multilevel security enforcement is mandatory and automatic.
- Multilevel security can use methods that are difficult to express through traditional SQL views or queries.
- Multilevel security can provide performance benefits with row-level checking.
- Multilevel security does not rely on special views or database variables to provide row-level security control.
- Multilevel security controls are consistent and integrated across the system, so that you can avoid defining users and authorizations more than once. Access to files, databases, printers, terminals, and other resources can have a single security control point.

Mandatory access control and dominance

In multilevel security, mandatory access control restricts access to an object based on the dominance relationships between user security labels and object security labels. One security label dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that defines one security label includes the set of security categories that defines the other security label.

Mandatory access control evaluates dominance and determines whether to allow certain actions, based on the following rules:

- If the security label of a user dominates the security label of an object, the user can read from the object.
- If the security label of a user and the security label of the object are equivalent, the user can read from and write to the object.
- If the security label of the object dominates the security label of the user or if the security labels are incompatible, the user cannot read from or write to the object.

Mandatory access control prevents users from declassifying information by not allowing a user to write to that object unless the security label of the user and the security label of the object are equivalent. You can override this security feature,

known as write-down control, for specific users by granting write-down authority to those users. The examples in the following section assume that the user does not have write-down authority.

Implementing and using multilevel security

This section briefly sketches how to implement multilevel security with row-level granularity on a table. For a complete discussion of how to implement multilevel level security with row-level granularity, see *z/OS Planning for Multilevel Security and the Common Criteria*. This section also demonstrates how multilevel security affects the results of SQL statements and utilities that LOAD, UNLOAD, and REORG DISCARD, and LOAD REPLACE rows.

Example: Suppose that you have a table EMP. Also, suppose that you need to implement mandatory and granular security to protect the sensitive data that is stored in EMP. To implement multilevel security with row-level granularity to protect the data in EMP, define a security label column. Define the security label column as CHAR(8) NOT NULL WITH DEFAULT and with the AS SECURITY LABEL clause, as shown in the following statement:

```
ALTER TABLE EMP
  ADD SECURITY   CHAR(8)       NOT NULL WITH DEFAULT AS SECURITY LABEL;
```

After the security label column is populated with security labels, DB2 enforces security checks for each row. These security checks affect the results of SELECT, INSERT, UPDATE, and DELETE statements, and utilities that load, unload, or delete rows.

Example: Suppose that Beth has the security label MEDIUM. Suppose that the table EMP contains the data that is shown in Table 11 and that the SECURITY column is the security label column.

Table 11. Sample data from EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
A00147	JONES	19	LOW
A00148	NGUYEN	19	HIGH
A00149	SANCHEZ	19	MEDIUM

Now, suppose that Beth issues the following statement:

```
SELECT LASTNAME
  FROM EMP
  ORDER BY LASTNAME;
```

Because Beth's security label MEDIUM dominates the security labels LOW and MEDIUM, she receives the following result:

```
JONES
SANCHEZ
```

Beth does not see NGUYEN in her result set because the row with that information has a security label of HIGH. Although Beth does not receive the full result set for her query, DB2 does not return an error code to Beth.

Example: Now, suppose that Beth issues the following statement on the EMP table:

```
UPDATE EMP
  SET WORKDEPT='17N'
  WHERE WORKDEPT='19' AND SECURITY=GETVARIABLE(SYSIBM.SECLABEL);
```

Because Beth has a security label that is equivalent to the security label of the row with MEDIUM security, that row is examined and the update succeeds for that row. Table 12 shows the results of the UPDATE statement.

Table 12. Sample data from EMP after the UPDATE statement

EMPNO	LASTNAME	WORKDEPT	SECURITY
A00147	JONES	19	LOW
A00148	NGUYEN	19	HIGH
A00149	SANCHEZ	17N	MEDIUM

SQL support for XML functions in DB2

Version 8 of DB2 UDB for z/OS provides a set of SQL built-in functions that allow applications to generate XML data from relational data. These functions reduce application development efforts for generating XML data for data integration, information exchange, and Web services. Version 8 includes the following XML functions:

- The XMLELEMENT function generates an XML element from a variable number of arguments.
- # The XMLNAMESPACES function generates XML namespace declarations.
- The XMLATTRIBUTES function constructs XML attributes from the arguments.
- The XMLFOREST function produces a forest of XML elements that all share a specific pattern from a list of columns and expressions. A *forest* is an ordered set of subtrees of XML nodes; XML nodes can represent an element, a text string, and so on.
- The XMLCONCAT function concatenates a variable number of arguments to generate a forest of XML elements.
- The XMLAGG function, an aggregate function, produces a forest of XML elements from a collection of XML elements.
- The XML2CLOB function converts the transient XML data type into a CLOB so that applications can access the XML data. The transient XML data type exists during query processing only.

If you plan to generate large XML documents by using the XML built-in functions (on the order of gigabytes, for example), your DB2 subsystem can consume a large amount of virtual storage space. In that case, you must configure the system to avoid performance degradation.

Example: Generate an ″Emp″ element for each employee. Use employee name as its attribute and two subelements generated from columns HIRE and DEPT by using XMLFOREST as its content. The element names for the two subelements are ″HIRE″ and ″department″.

```
SELECT e.id, XML2CLOB ( XMLELEMENT
  ( NAME "Emp",
  XMLATTRIBUTES ( e.fname || ' '  || e.lname AS "name" ),
  XMLFOREST ( e.hire, e.dept AS "department" ) ) ) AS "result"
    FROM employees e;
```

The result of the query would be similar to the following result:

```
ID     result
---------------------------------------------
1001   <Emp name="John Smith">
         <HIRE>2000-05-24</HIRE>
         <department>Accounting</department>
```

```
          </Emp>
1001   <Emp name="Mary Martin">
         <HIRE>1996-02-01</HIRE>
         <department>Shipping</department>
          </Emp>
```

Example: Concatenate first name and last name elements by using “first” and “last” element names for each employee.

```
SELECT XML2CLOB( XMLCONCAT
  ( XMLELEMENT ( NAME "first", e.fname),
  XMLELEMENT ( NAME "last", e.lname ) ) ) AS "result"
     FROM employees e;
```

The result of the query would look similar to the following result, where the ″result column″ is a CLOB:

```
result
 ---------------------------------------------
 <first>John</first><last>Smith</last>
 <first>Mary</first><last>Smith</last>
```

For more information about XML publishing functions, see *DB2 SQL Reference*.

Improvements in connectivity

Version 8 of DB2 includes the following connectivity enhancements:

- “Enhanced support for JDBC and CLI clients”
- “Easier access to remote workstation database through database alias support” on page 62
- “More granular control of routing requests to specific members of a data sharing group” on page 62
- “Improved JDBC and CLI connectivity for cursors and result sets” on page 63
- “More flexibility in managing distributed applications with CURRENT PACKAGE PATH special register” on page 63

Enhanced support for JDBC and CLI clients

In versions of DB2 before Version 8, different connection protocols exist for access to a DB2 UDB for Linux, UNIX®, and Windows® server and for access to a DB2 UDB for z/OS server. Each server protocol uses a different set of methods to implement the same functions. To reduce complexity and duplication, Version 8 of DB2 provides access across the DB2 UDB family by implementing the DRDA standard that is published by The Open Group. This standard defines an open, published architecture for communication among applications, application servers, and database servers on platforms with the same or different hardware and software architectures.

The Open Group DRDA Standard supports functions that you can implement on the Linux, UNIX, and Windows platforms and the z/OS server. These standards are described in the Open Group Technical Standard DRDA Version 3.

For Version 8, DB2 support includes new server functions for specific C-based common client requirements, Java-based common client requirements, and iSeries™ requirements that are not supported for z/OS applications. It provides an open and consistent set of protocols for the different platforms on which the data resides.

The new features include:

- **Cursor and stored procedure result set instances:** A DB2 UDB for z/OS server now allows multiple instances of a cursor, or multiple stored procedure result sets, to be open concurrently under the same thread.
- **Extended describe:** A DB2 UDB for z/OS server can provide extended descriptive information to support the JDBC 2.0 updateRow and deleteRow methods.
- **SQL cancel:** A JDBC or CLI application can cancel long-running requests on a DB2 UDB for z/OS server.
- **Cursor extensions:** DB2 UDB for z/OS allows a requester to identify:
  - Whether the server should release read locks when a query is closed.
  - Whether the server should close a query implicitly when no more rows exist for a non-scrollable cursor, regardless of whether the cursor has the HOLD attribute.
- **Better utilization of network capacity:** DB2 UDB for z/OS provides more flexibility for requesters such as DB2 Connect™ to specify larger query block sizes. This helps requesters optimize their use of network resources.
- **Requester database aliases:** A database administrator can specify multiple locations for DB2 UDB for Linux, UNIX, and Windows databases.
- **Distributed transactions:** DB2 UDB for z/OS, Version 8 adds DRDA XA protocol support, which is needed to support Java Transaction API (JTA)/Java Transaction Service (JTS) distributed transactions. This support is available only for TCP/IP connections.
- **Server location aliases:** DB2 UDB for z/OS supports location aliases that reflect the location names used by applications to route requests to all or a subset of members in a data sharing group.
- **Subsets:** DB2 UDB for z/OS allows you to define subsets of data sharing group members in TCP/IP networks. A non-DB2 UDB for z/OS requester can connect to a subset of data sharing group members by appending a port number to a location alias.
- **Member routing in a TCP/IP network:** In data sharing environments, applications can route requests directly to one or more members of a data sharing group. They do so by using location names on the client side in conjunction with location aliases on the server side. The location names can represent a specific member, multiple members, or all members of a data sharing group.
- **Timeout for allocate conversation requests:** If a VTAM® request to allocate a conversation for a remote SQL statement does not complete in three minutes, DDF forces VTAM to abnormally terminate the remote request .

Easier access to remote workstation database through database alias support

The communications database allows a DB2 requester to access multiple DB2 UDB for Linux, UNIX, and Windows databases that are set up with the same database name. DB2 provides the use of a database alias (DBALIAS) in SYSIBM.LOCATIONS to override the location name that a z/OS application uses to access a server.

More granular control of routing requests to specific members of a data sharing group

Implementing member routing in TCP/IP networks requires that a site define both client-side location names and server-side location aliases. Requesters use the

SYSIBM.IPLIST table to define location names that represent the members to which requests are to be routed. A requester can define multiple location names, each of which represents a different subset of the members of the data sharing group. The SYSIBM.IPLIST table maps location names to member-specific domain names. On the server side, the DB2 administrator uses the ALIAS option of the DSNJU003 (change log inventory) utility to update the bootstrap data set (BSDS) with location aliases. Location aliases identify location names that are used by requesters to access members of the data sharing group.

Improved JDBC and CLI connectivity for cursors and result sets

Before Version 8 of DB2, the second open of a cursor always failed if the cursor was already open. The second call to the same stored procedure always closed any open result sets. A requester can now open the same cursor multiple times, or it can process result sets from different calls to the same stored procedure. A DB2 UDB for z/OS server provides a unique identifier to the requester for each open cursor or result set. The requester can then manage the multiple instances using the unique cursor identifier.

For example, before Version 8 of DB2, customizing SQLJ applications to call the same method twice was difficult. In Version 8, the requester can determine if multiple instances of the cursor need to be generated. This allows the requester to manage the different cursor instances.

More flexibility in managing distributed applications with CURRENT PACKAGE PATH special register

Package collections let you logically group packages for general administration or housekeeping, and provide a way to maintain different versions of an application. You can bind a package into multiple collections, and you need a convenient way to search the collections for a specific package. The new CURRENT PACKAGE PATH special register lets you specify a list of collections in which to search for a package. The SET CURRENT PACKAGE PATH SQL statement is similar to the PKLIST bind option, but the SET CURRENT PACKAGE PATH statement is processed at the server.

In releases of DB2 before Version 8, the only way to switch between packages was to execute the SET CURRENT PACKAGESET statement every time you needed to use a different package. With SET CURRENT PACKAGE PATH, you can execute the statement only once, to give the server a list of package collections to search.

CURRENT PACKAGE PATH is especially important for Java applications that use SQLJ. SQLJ applications are written in Java, so you can run them on a variety of platforms. However, different database servers support different sets of bind options. You therefore need to bind the same program into several packages, each with a different collection ID, and each with a different set of bind options. In the program, you can execute one SET CURRENT PACKAGE PATH statement to list the collections that are to be searched at all database servers. When the program connects to a server, the server locates the package with the associated collection ID to run.

Other e-business enhancements

Version 8 of DB2 UDB for z/OS introduces the following additional e-business enhancements:

- SQL processing options CCSID and NEWFUN; see "SQL processing options."
- Resource Recovery Services attachment facility (RRSAF) implicit connections to DB2; see "RRSAF implicit connections."
- Multiple instances of the same stored procedure can be run concurrently. See "Changes to stored procedures processing."
- Support for 31-digit DB2 PL/I applications; see "Enhancements for DB2 PL/I applications" on page 65.

SQL processing options

The SQL processing option CCSID(*n*) specifies the numeric value *n* of the CCSID in which the source program is written. The default setting is the EBCDIC system CCSID as specified on the panel DSNTIPF during installation.

The SQL processing option NEWFUN indicates whether to accept the syntax for DB2 Version 8 functions. NEWFUN(YES) causes the precompiler to accept Version 8 syntax. A successful precompilation produces a DBRM that can be bound only with Version 8 and later releases, even if the DBRM does not use any Version 8 syntax. NEWFUN(NO) causes the precompiler to reject any syntax that DB2 Version 8 introduces. A successful precompilation produces a DBRM that can be bound with any release of DB2, including Version 8.

During migration to Version 8 from Version 7, the NEWFUN default value is NO. At the end of enabling-new-function mode, the default changes from NO to YES.

If Version 8 is a new installation of DB2, the default is YES.

RRSAF implicit connections

If you do not explicitly specify the IDENTIFY function in a CALL DSNRLI statement, RRSAF initiates an implicit connection to DB2 if the application includes SQL statements or IFI calls. An implicit connection causes RRSAF to initiate implicit IDENTIFY and CREATE THREAD requests to DB2. Although RRSAF performs the implicit connection request by using default values, the request is subject to the same DB2 return and reason codes as are explicit connection requests.

For an implicit connection request, your application should not specify either IDENTIFY or CREATE THREAD. However, an implicit connection does not perform any SIGNON processing. Your application can execute SIGNON at any point of consistency and any other RRSAF calls after the implicit connection. To terminate an implicit connection, you must use the proper calls.

Your application program must successfully connect, either implicitly or explicitly, to DB2 before it can execute any SQL calls to the RRSAF DSNHLI entry point.

Changes to stored procedures processing

Your application program can issue multiple CALL statements to the same local or remote stored procedure. If that stored procedure returns result sets and the calling application leaves those result sets open before the next call to that same stored procedure, each CALL statement invokes a unique instance of the stored procedure. Each instance of the stored procedure 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.

In compatibility mode and enabling-new-function mode, multiple calls to the same stored procedure do not produce multiple instances of the applications.

To invoke multiple instances of remote stored procedures or local stored procedures that have SQL to access a remote site, both the client and server must be in DB2 Version 8 new-function mode or later. For local stored procedures that issue remote SQL, instances of the applications are created at the remote server site regardless of whether result sets exist or are left open between calls.

DB2 storage shortages and EDM POOL FULL conditions can occur if you call too many instances of a stored procedure or if you open too many cursors. 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.

To optimize storage usage, two subsystem parameters control the maximum number of stored procedures instances and the maximum number of open cursors for a thread. 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 open 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.

The calling application for the stored procedure should close the result sets and issue commits often. Even read-only applications should perform these actions. Applications that fail to do so terminate abnormally with DB2 storage shortage and EDM POOL FULL conditions.

You can set the maximum number of stored procedure instances and the maximum number of open cursors on installation panel DSNTIPX. For more information about setting the maximum number of stored procedure instances and the maximum number of open cursors per DB2 thread or connection, see the topic "Routine parameters panel: DSNTIPX"in *DB2 Installation Guide*.

Enhancements for DB2 PL/I applications

DB2 provides support for 31-digit precision in DB2 PL/I applications. To use 31-bit precision in your applications, you must have VisualAge PL/I for OS/390, Version 2 Release 2 or later.

Chapter 3. Planning for migration, conversion, and fallback

This chapter contains considerations for migration, for conversion to new-function mode, and for fallback from compatibility mode to Version 7. A directory of new and revised installation panels is also provided in this chapter. You can migrate to Version 8 compatibility mode only from Version 7. See *DB2 Installation Guide* for complete, step-by-step instructions for installing, migrating, converting to new-function mode, or falling back.

Migration to Version 8 is comprised of three progressive catalog levels: compatibility mode, enabling-new-function mode, and new-function mode.

Compatibility mode
The state of the catalog after the Version 8 migration process is complete.

Enabling-new-function mode
Marked by the beginning and ending of catalog conversion job DSNTIJNE, which converts catalog data to Unicode.

New-function mode
Begins after a the successful completion of job DSNTIJNF.

The following topics provide additional information:
- "Hardware and software requirements"
- "Migration considerations"
- "Preparing for fallback" on page 80
- "Release incompatibilities" on page 82
- "Release coexistence" on page 87
- "Installation changes" on page 88

Hardware and software requirements

Detailed information about hardware and software requirements for Version 8 of DB2 UDB for z/OS can be found in the Version 8 *DB2 Program Directory*. This book is shipped with the product and is available on the Web at www.ibm.com/software/db2zos/library.html.

Migration considerations

This section includes items to consider before migrating DB2 to Version 8 compatibility mode from Version 7.

Make sure that your current subsystem is at the proper service level. Before you migrate to Version 8 compatibility mode, you must have a maintenance level on Version 7 that contains the fallback SPE. If you do not have the fallback SPE applied, your Version 8 compatibility mode migration process terminates. See the Version 8 *IBM DB2 Program Directory*, which is shipped with the product, for keyword specifications for preventive service planning (PSP). Check Information/Access or the ServiceLink facility of IBMLink™ for PSP information before you migrate. Also check those facilities monthly to obtain the most current information about DB2.

You must migrate to a z/OS Version 1 Release 3 environment (or later) before migrating to Version 8 compatibility mode. Other facilities, such as CICS, IMS, and COBOL also must be migrated to later releases. See *DB2 Installation Guide* for more details.

DB2 Version 8 publications assume new-function mode

All publications in the DB2 Version 8 library assume that your DB2 subsystem is running in Version 8 new-function mode. Changes to functions, statements, and limits are available in new-function mode unless stated otherwise.

DB2 Utilities Suite for z/OS Version 8 uses the DFSORT program

The DB2 Utilities Suite for z/OS Version 8 is designed to work with the DFSORT program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES
- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

Use triggers instead of field, edit, and validation procedures

It is recommended that you use triggers instead of field, edit, and validation procedures. Triggers can have long names, but field, edit, and validation procedures are limited to names of 18 bytes or smaller.

DB2 treats certain large fixed-length strings as varying-length strings

Fixed-length character host variables cannot be over 255 bytes long. Fixed length graphic host variables cannot be over 127 characters long. When DB2 manipulates a fixed-length character string that is declared as more than 255 bytes long, or a fixed-length graphic string that is declared as more than 127 characters long, DB2 treats that string as a varying-length string.

MEMLIMIT cannot be customized through the installation process

In previous releases of DB2, you could modify the MEMLIMIT setting for the DBM1 address space by setting a value for STG AVAILABLE ABOVE 2GB in panel DSNTIPC. This field has been removed and DB2 now determines the appropriate setting.

DBDs cannot be accessed if DB2 starts in deferred mode

If you start DB2 in a deferred mode, database descriptors (DBDs) cannot be accessed until the restart has completed. If you attempt to load a DBD during DB2 start-up in deferred mode, the DBD is not loaded and DB2 start-up continues.

DB2 LOCATION NAME value

In previous releases of DB2, DB2 could start if a value was not specified in the DB2 LOCATION NAME field in panel DSNTIPR. However, in DB2 Version 8, DB2 will not start if no value has been specified in that field.

Type 1 indexes are not supported

Before you migrate to Version 8, you must convert all type 1 indexes to type 2 indexes. DB2 migration fails if your subsystem contains type 1 indexes.

Declared global temporary tables need an 8-KB buffer pool

Global temporary tables require an 8-KB buffer pool, which are required to install DB2. Existing jobs that create a table space in the temporary database might also need to be modified.

Declared global temporary tables need an 8-KB table space in the temporary database

If you use declared temporary tables, you must define at least one of the table spaces in the temporary database to have a page size of 8 KB or greater. Member DSNTESQ of the *prefix*.SDSNSAMP library contains a sample query to check your temporary databases.

System-level point-in-time recovery

If you plan to use the BACKUP SYSTEM online utility to take volume copies of the data and logs of your non-data-sharing DB2 subsystem or of your DB2 data sharing group, all of your DB2 data sets must reside on volumes that are managed by DFSMS. The BACKUP SYSTEM utility and its counterpart, the RESTORE SYSTEM utility, require:

- z/OS Version 1 Release 5 or above.
- Disk control units that support ESS FlashCopy®.
- HSM copy pools whose definitions follow the DB2 naming convention.
- SMS copy target storage pools that are defined. (The BACKUP SYSTEM utility enables volume-level backups of a DB2 subsystem that uses these target storage groups.)

Exception: If you use RESTORE SYSTEM with the LOGONLY option, you do not need the preceding requirements. You can perform the restoration manually by using your preferred method, and then run RESTORE to complete the recovery.

Enhanced support for scrollable cursors

Support for scrollable cursors enables dynamic access to data in tables. In Version 7, scrollable cursors required storage space in the temporary database and in segmented table spaces. In Version 8, with dynamic scrollable cursors, this restriction no longer exists, which might result in a decrease in the needed storage.

Changes to space allocations for DB2-managed data sets

The default values for primary space allocations have increased. For non-LOB table spaces and indexes, the default primary space allocation is one cylinder. For LOB table spaces, the default primary space allocation is ten cylinders.

The default values for secondary space allocations can now use a sliding scale. If you specify a value of YES for the field OPTIMIZE EXTENT SIZING on panel DSNTIP7, DB2 uses a sliding scale to determine the secondary allocations for DB2-managed data sets if you explicitly specify SECQTY (with a valid value that is not -1) in the CREATE TABLESPACE or CREATE INDEX statement or in any of the subsequent ALTER TABLESPACE or ALTER INDEX statements.

Using a sliding scale for secondary space allocations might result in increased disk space usage. However, overall, this method generally results in better space utilization and fewer situations in which the maximum number of extents are reached.

Changed default value for DESCRIBE FOR STATIC

During installation of DB2 Version 8, the default for subsystem parameter DESCSTAT on installation panel DSNTIP4 is now YES. If your DB2 UDB for z/OS subsystem or DB2 UDB for Linux, UNIX, and Windows systems uses the new JDBC driver, or if your DB2 UDB Linux, UNIX, and Windows systems uses the new CLI driver, you must set DESCSTAT to YES.

Changed data types and lengths for some catalog columns

Some catalog columns have new data types and lengths. In Version 8, they are now VARCHAR(*n*), where *n* is 8 or greater.

If your application program uses the values of these columns in comparison statements such as a statement that uses a LIKE predicate, you might need to adjust your application program to get the desired results.

Queries that contain an IN or NOT IN subquery can no longer use a sparse index
because sparse index does not support VARCHAR.

Changed data types and lengths for some special registers

Some special registers have new data types and lengths. The changed registers and their new data types and lengths are:

- CURRENT OPTIMIZATION HINT is now VARCHAR(128).
- CURRENT PACKAGESET is now VARCHAR(128).
- CURRENT SQLID is now VARCHAR(8).
- USER is now VARCHAR(8).
- CURRENT PATH is now VARCHAR(2048).

If your application program uses the values of these registers in comparison statements such as a LIKE predicate, you might need to adjust your application program to get the desired results.

SQL reserved words may be used in delimited identifiers for procedure names

In Version 8, you may use SQL reserved words in delimited identifiers for procedure names. See DB2 SQL Reference for more information.

Encoding schemes of string parameters for routines

The new PARAMETER CCSID clause allows you to define the encoding scheme of all string parameters for user-defined functions and stored procedures at the same time. In previous versions, you needed to define a CCSID for each string parameter if you wanted an encoding scheme other than the default. Also, EBCDIC is no longer the default encoding scheme for system-defined parameters. DB2 now uses the same encoding scheme for both user-specified and system-generated string parameters.

Modify RUNSTATS jobs

After you migrate to Version 8, some existing RUNSTATS jobs might fail if
data-partitioned secondary indexes are defined on the tables on which they run.

RUNSTATS jobs on data-partitioned secondary indexes require sort operations; therefore, you need to modify these RUNSTATS jobs to allocate the sort work data sets. You can modify the RUNSTATS jobs with the SORTDEVT and SORTNUM keywords, or you can add STATWK*nn* DD statements to the JCL.

More history statistics are collected

If you specify SPACE or ACCESSPATH for the STATISTICS HISTORY parameter on panel DSNTIPO, DB2 might insert more statistics into the catalog statistics history tables. For example, DB2 inserts statistics when you run a utility with the UPDATE(ACCESSPATH) or UPDATE(SPACE) parameter but without the HISTORY parameter.

Creating tables with DBCS and mixed columns

You can no longer create extended binary-coded decimal interchange code (EBCDIC) tables with GRAPHIC, VARGRAPHIC, or DBCLOB columns when the setting for installation option MIXED DATA is NO. You also cannot alter EBCDIC tables to add GRAPHIC, VARGRAPHIC, or DBCLOB columns when MIXED DATA is NO.

Consider increasing IDBACK and CTHREAD

Because utilities might use additional threads, you should consider increasing the values of the IDBACK and CTHREAD subsystem parameters. Increasing these parameter values can help you avoid failure of some utilities due to increased thread usage. An increase also supports the additional parallelism that is associated with the utilities.

Support for DB2-established data space for cached dynamic statements is deprecated

In Version 8, support for a DB2-established data space for cached dynamic statements is deprecated. You can no longer specify the parameters EDMDSPAC or EDMDSMAX during installation or migration. A new EDM statement cache is provided for cached dynamic statements. See *DB2 Installation Guide* for a description of the parameters for the new EDM statement cache.

Consider changing EDM pool size

Cached dynamic statements and database descriptors are in a separate pool in Version 8, which could result in decreased storage requirements. You can change the EDM pool size by modifying the EDMPOOL STORAGE SIZE field on installation panel DSNTIPC, and then stopping and restarting DB2. You can also modify the EDM pool size without stopping and restarting DB2 by using the SET SYSPARM command. However, using the SET SYSPARM command might result in a pool that is not contiguous, which is less efficient.

Customized DB2I defaults can be migrated

You can migrate a DB2I TSO IPSF profile member from a prior release to the current release. The DSNEMC01 CLIST uses the values that are specified on installation panel DSNTIPF and stores the results in the ISPF profile member DSNEPROF. You can migrate any customized DSNEPROF members from Version 7 to Version 8. However, you need to examine any new or changed default panel values to ensure that your customized values are still valid.

Rebinding DSNACOLN and the DatabaseMetadata stored procedures (for ODBC and JDBC support)

If you rebind all packages after migrating from DB2 Version 7 to DB2 Version 8 compatibility mode, the DSNACOLN metadata package from Version 7 is included, even though it is no longer valid for Version 8. The DSNACOLN metadata package from Version 7 is only intended to run in DB2 Version 7 environments and these provided DBRMs are no longer needed following a successful migration, unless you fall back to Version 7. Rebinding the DSNACOLN metadata package from Version 7 while running Version 8 compatibility mode can result in the following abend:

```
DUMP TITLE=DSN ,ABND=04E-00E70005,U=xxxxxx  ,M=(C),C=810.SCC
               -REBNDPKG,M=DSNTFRCV,LOC=DSNXGRDS.DSNHPARS:P502
```

After you migrate from DB2 Version 7 to DB2 Version 8, use the provided Version 8 DBRMs instead of the previous ones that were provided for Version 7. The Version 8 DBRMs are provided for both Version 8 compatibility mode and Version 8 new function mode. These packages are precompiled and contain the necessary CCSID changes for Version 8.

Furthermore, after you migrate to Version 8, if you use a multi-byte EBCDIC character set, you might encounter the above ABEND when you attempt to rebind packages in collection ID DSNASPCC that were created in DB2 Version 7 or an earlier release. DSNASPCC is the collection id for stored procedures that support the JDBC and ODBC DatabaseMetadata stored procedures (SYSIBM.SQL*). Use the BIND PACKAGE statements for DSNASPCC in your customized copy of installation job DSNTIJSG to create fresh packages for the JDBC and ODBC DatabaseMetadata stored procedures while DB2 is running in Version 8 compatibility mode. When DB2 enters Version 8 new function mode, use the BIND PACKAGE statements for DSNASPCC in job DSNTIJMC.

LANGUAGE COMPJAVA no longer supported for stored procedures

After migrating to Version 8, you can no longer define or run COMPJAVA stored procedures. Convert LANGUAGE COMPJAVA stored procedures to LANGUAGE JAVA by following these steps:

1. Use ALTER PROCEDURE to change the LANGUAGE and the WLM ENVIRONMENT. The EXTERNAL NAME clause must also be specified. Use the following example as a model:

   ```
   ALTER PROCEDURE SYSPROC.JAVADVR
   LANGUAGE JAVA EXTERNAL
   NAME 'display.display.main'
   WLM ENVIRONMENT WLMENVJ;
   ```

   You must specify a valid language option when issuing any ALTER PROCEDURE statement for a procedure that was defined with LANGUAGE COMPJAVA. If you do not, DB2 issues an error.
2. Ensure that the WLM environment is configured and that the required JVM is installed.
3. Ensure that the .class file that is identified in the EXTERNAL NAME clause of the ALTER PROCEDURE is present in one of the following places:
   - In a JAR that was installed to DB2 by an invocation of the INSTALL_JAR stored procedure

- In a directory in the CLASSPATH ENVAR of the data set that is named on the JAVAENV DD statement of the WLM stored procedures address space JCL

DSNWZP runs in WLM-established stored procedure address space

In DB2 Version 8, the DB2-supplied stored procedure DSNWZP is defined to run in a WLM-established stored procedure address space that uses external module DSNWZP. If you ran DSNWZP in a WLM-established stored procedure address space in DB2 Version 7, you redefined DSNWZP to use external module DSNWZPR. If you do not use job DSNTIJSG to define DB2-supplied stored procedures in DB2 Version 8, you must alter stored procedure DSNWZP to use external module DSNWZP.

Support for DB2-established stored procedure address spaces is deprecated

In Version 8, support for DB2-established address spaces is deprecated. You can no longer specify the NO WLM ENVIRONMENT option when you create or alter stored procedure definitions. Although existing stored procedures can still run in a DB2-established stored procedure address space, you should move your stored procedures to WLM environments as soon as possible. For more information about moving stored procedures, see Part 5 (Volume 2) of *DB2 Administration Guide*.

Pre-compilation for unsupported compilers

For some COBOL and PL/I compilers that are no longer supported by Version 8, you can use a version of the precompiler that allows you to precompile applications that have dependencies on these unsupported compilers. You can use this version of the precompiler with the following unsupported compilers:

- OS/VS COBOL V1.2.4
- OS PL/I 1.5 (PL/I Opt. V1.5.1)
- VS/COBOL II V1R4
- OS PL/I 2.3

The load module for this precompiler is DSNHPC7. This precompiler is meant only to ease the transition from unsupported compilers to supported compilers. This precompiler has the following restrictions:

- It is available only for Version 8.
- There is no corresponding DB2 coprocessor function to match this precompiler.
- The precompiler does not support SQL procedures.
- Only COBOL and PL/I are supported.
- The SQL flagger is not supported.
- The precompiler produces Version 7 DBRMs, and therefore does not support any capability that is new to Version 8.

Use this version of the precompiler only until you migrate your applications to work with supported compilers.

New precompiler option for string host variables

In previous releases of DB2, if you selected a value from a character column into a C or C++ host variable of the nul-terminated character form, and the length of the host variable was longer than the length of the value, DB2 padded the string with

blanks and inserted the nul-terminator after the blanks. In Version 8, the DB2 default behavior is to not pad the string with blanks. If you want to produce blank-padded strings, as in previous releases, specify YES in field PAD NUL-TERMINATED in installation panel DSNTIP4, or precompile your program with the PADNTSTR option.

You must specify the APOST precompiler option when the given CCSID for the source is 1026 or 1155

You cannot specify the following precompiler options together:

- CCSID(1026) or CCSID(1155)
- HOST(IBMCOB)
- QUOTE

When you precompile an IBM COBOL program, and the source CCSID is 1026 or 1155, you need to specify APOST instead of QUOTE.

New SYSIBM.SYSROUTINES column for encoding scheme

After you successfully migrate to Version 8, the encoding scheme that is used for system-generated parameters for procedures and functions is stored in a new column in SYSIBM.SYSROUTINES. This information was previously stored in a special row in the SYSIBM.SYSPARMS table.

LANGUAGE REXX sets PROGRAM_TYPE column in SYSIBM.SYSROUTINES

If you specify LANGUAGE REXX, DB2 sets the PROGRAM_TYPE column in SYSIBM.SYSROUTINES to 'M'. You cannot override this value by specifying PROGRAM TYPE MAIN or PROGRAM TYPE SUB. The procedure will continue to run as in Version 7, where all REXX procedures were treated as a main procedure.

DB2 start-up and precompilation require a user-supplied DSNHDECP module

Installation job DSNTIJUZ generates the data-only load module DSNHDECP. It contains the application programming defaults. DB2 is shipped with a default DSNHDECP for compatibility with older applications. You cannot start DB2 or precompile applications with the default DSNHDECP. During DB2 start-up processing or for jobs that precompile a DB2 application, a DSNHDECP module that is customized by the installation CLIST must exist in a library that is before the library that contains the default DSNHDECP module in the STEPLIB concatenation, the JOBLIB concatenation, or the system link list.

CCSIDs in DSNHDECP must be valid

All CCSIDs in the DSNHDECP module must be valid. During start-up processing, if DB2 detects invalid CCSID values, DB2 issues a message and terminates.

Character conversions between Unicode CCSIDs and EBCDIC CCSIDs

The character conversions between Unicode CCSIDs 367, 1208, and 1200, and EBCDIC CCSIDs 37, 500, and 1047 must be defined.

New data-only load module DSNHMCID

The new data-only load module DSNHMCID contains EBCDIC CCSIDs for offline message conversion. Version 8 utilities and applications must have access to this module. You can provide access to DSNHMCID in one of the following ways:

- Permit the DSNHMCID module to reside in DSNSLOAD.
- Include the library SDSNEXIT before SDSNLOAD in the system link list.
- Verify that all jobs and tasks that use DB2 utilities or call DB2 application programs are updated to STEPLIB or JOBLIB to SDSNEXIT.

Plans and packages bound prior to DB2 Version 2 Release 3

If you have plans and packages that were bound prior to DB2 Version 2 Release 3, DB2 will autobind these packages. Thus, you may experience an execution delay the first time that such a plan is loaded. Also, DB2 may change the access path due to the autobind, potentially resulting in a more efficient access path.

Multiple calls to the same stored procedure

In previous versions of DB2, if a stored procedure was called twice from the same program and at the same nesting level, DB2 closed the result set cursors and released storage for the first instance of the stored procedure before making the second call. In DB2 Version 8, if the requester and the server are both DB2 Version 8 subsystems in new-function mode, when the second call is made, both instances of the stored procedure can run at the same time. DB2 does not close the result sets from the first call or release storage for the first instance of the stored procedure.

External stored procedures and user-defined functions can return any valid SQLSTATE value

In previous versions of DB2, an external stored procedure or user-defined function could return only SQLSTATE values of the form '01Hxx', '38xxx', '00000', or '02000'. In DB2 Version 8, an external stored procedure or user-defined function can return any valid SQLSTATE value.

Programs called by a stored procedure require packages

In previous versions of DB2, if a stored procedure called a subprogram using a host language call, and that subprogram contained SQL statements, DB2 did not require a package for that subprogram at the location where the stored procedure was defined. In DB2 Version 8, if a stored procedure calls a subprogram that contains SQL statements, and a package does not exist for that subprogram at the server where the stored procedure is defined, DB2 issues an error message.

Port of entry name changed

If you are using z/OS Version 1 Release 5, TCP/IP Network Access Control, and the RACF SERVAUTH class is active, the port of entry name that is passed to RACF for verification is the point of entry security zone name. The port of entry security zone name is defined in the TCP/IP Network Access Control profile. In previous releases of DB2, the port of entry name that was passed to RACF was the string 'TCPIP'.

New name for type 1 inactive threads and type 2 inactive threads

Type 1 inactive threads are now referred to as inactive DBATs. Type 2 inactive threads are now referred to as inactive connections.

Column names and labels in SQLDA SQLNAME field for statements involving UNION

Prior to Version 8, the result column name in a SQLNAME field of the SQLDA for a statement involving a UNION reflected the column name or label of the first sub-query in the statement. In Version 8, DB2 returns the name or the label of the column only if the name or label is the same for that column across all sub-queries in the statement. Otherwise, the result column name will be blank.You can temporarily override this behavior by setting subsystem parameter UNION_COLNAME_7 to YES.

MAXROWS must have a value of 1 on ALTER TABLESPACE DSNDB06.SYSSEQ

You can no longer specify any MAXROWS value except MAXROWS 1 on ALTER TABLESPACE DSNDB06.SYSSEQ.

IFCID 197 is no longer supported

In Version 8, IFCID 197 is no longer supported. If you make a READS call for IFCID 197, DB2 issues return code 8 and reason code 00E60821.

Change to IFCID 0059 trace records

IFCID 0059 records the start of execution of an SQL FETCH statement. In Version 7, one IFCID 0059 trace record is generated for each row that is fetched. However, in Version 8, if limited block fetch is used, one IFCID 0059 record is generated for each block that is fetched.

Change data capture cannot be enabled on catalog tables during enabling-new-function mode

During enabling-new-function mode processing, change data capture is disabled on most catalog tables. You cannot re-enable change data capture until your DB2 subsystem is in Version 8 new-function mode.

DB2 Version 8 requires IRLM 2.2

IRLM 2.2 is delivered with DB2 Version 8. You must use the DB2–supplied IRLM procedure.

Detailed tracking of DB2 measured usage is disabled

In previous releases of DB2, DB2 automatically used detailed tracking of measured usage. In Version 8, subsystem parameter SMF89 controls whether DB2 uses detailed tracking of measured usage. The default value is NO, which means that DB2 does not do detailed measured usage tracking. If the SMF type 89 record is activated, only high-level tracking is recorded in the SMF type 89 record.

Programming language support has changed

Programming language support in DB2 Version 8 has changed. For a list of all supported languages, see *DB2 Program Directory*. If your DB2 Version 7 subsystem

uses languages other than those specified in *DB2 Program Directory*, you must migrate to a supported release of that language before migrating your DB2 subsystem to Version 8.

New return code for -START DATABASE, -STOP DATABASE, and -DISPLAY DATABASE commands

In previous releases of DB2, if the object of a -START DATABASE, -STOP DATABASE, or -DISPLAY DATABASE command was not found, the command completed with a return code of 12. In Version 8, if the object of a -START DATABASE, -STOP DATABASE, or -DISPLAY DATABASE command is not found, the command completes with a return code of 0. The behavior of these three commands is now similar to the behavior of other commands.

Views might be marked with view regeneration errors

DB2 automatically regenerates views that reference the DB2 catalog. However, as a result of changes to the catalog, some views may be marked with view regeneration errors. Views that are marked with view regeneration errors may be usable, but will not be automatically regenerated. You must manually regenerate these views.

Changed default values for subsystem parameters

The default values for several parameters have changed. The new values are listed in Table 13.

Table 13. Subsystem parameters with new default values

Panel	Field	Parameter	Version 7 default value	Version 8 default value
DSNTIP7	USER LOB VALUE STORAGE	LOBVALA	2048	10240
DSNTIPE	MAX USERS	CTHREAD	70	200
	MAX REMOTE ACTIVE	MAXDBAT	64	200
	MAX REMOTE CONNECTED	CONDBAT	64	10000
	MAX TSO CONNECT	IDFORE	40	50
	MAX BATCH CONNECT	IDBACK	20	50
DSNTIPN	DDF/RRSAF ACCUM	ACCUMACC	NO	10
DSNTIP8	CACHE DYNAMIC SQL	CACHEDYN	NO	YES
DSNTIPP	PLAN AUTH CACHE	AUTHCACH	1024	3072
DSNTIPL	LOG APPLY STORAGE	LOGAPSTG	0	100
	CHECKPOINT FREQ	CHKFREQ	50000	500000
DSNTIPA	BLOCK SIZE	BLKSIZE	28672	24576
DSNTIPR	DDF THREADS	CMTSTAT	ACTIVE	INACTIVE
	IDLE THREAD TIMEOUT	IDTHTOIN	0	120
	EXTENDED SECURITY	EXTSEC	NO	YES
DSNTIP5	TCP/IP KEEPALIVE	TCPKPALV	ENABLE	120
DSNTIPC	MAXIMUM OPEN DATA SETS	DSMAX	3000	10000
	EDMPOOL STORAGE SIZE	EDMPOOL	7312	32768[1]

Note:

[1] The installation CLIST calculates the default value for the EDMPOOL parameter.

If you specify SMFSTAT=YES, DB2 starts traces for SMF classes 1, 3, 4, 5, and 6. In DB2 Version 7, specifying SMFSTAT=YES only started traces for SMF classes 1, 3, 4, and 5.

If the values that you specified for these parameters are lower than the new default values, you might want to increase your values.

Subsystem parameter CLAIMDTA has been removed

Subsystem parameter CLAIMDTA has been removed. In Version 8, DB2 always operates as if CLAIMDTA=YES.

DSN8EXP is deprecated

DSN8EXP is deprecated and will be removed in a future release. Use DSNAEXP as your EXPLAIN stored procedure.

Using ALTER TABLE ALTER COLUMN SET DATA TYPE in compatibility mode places indexes in rebuild-pending state

In Version 8 compatibility mode, if ALTER TABLE ALTER COLUMN SET DATA TYPE is used to change the length of a VARCHAR column and that column is part of an index, the index is placed in rebuild-pending state.

Redundant DISTINCT keyword removed from SELECT DISTINCT statements

Starting in Version 8 compatibility mode, DB2 is enhanced to remove the DISTINCT keyword from SELECT DISTINCT statements to avoid unnecessary sort if the DISTINCT keyword is redundant. For example, the following SELECT statement contains a redundant DISTINCT keyword:

```
SELECT DISTINCT C1 FROM T1 GROUP BY C1
```

In Version 8 compatibility mode, the statement is rewritten by removing the redundant DISTINCT keyword as follows:

```
SELECT C1 FROM T1 GROUP BY C1
```

This enhancement can cause an incompatible difference in behavior between Version 7 and Version 8 compatibility mode for those SQL statements that are invalid in Version 7 only because of a redundant DISTINCT keyword in the SELECT DISTINCT statement. For example, the following SELECT DISTINCT statement receives SQLCODE -127 in Version 7:

```
SELECT DISTINCT C1, COUNT(DISTINCT C2) FROM T1 GROUP BY C1
```

Beginning in Version 8 compatibility mode, the statement is rewritten by removing the redundant DISTINCT keyword as follows:

```
SELECT C1, COUNT(DISTINCT C2) FROM T1 GROUP BY C1
```

The revised statement receives SQLCODE 0.

DB2 issues an error for column names greater than 30 bytes

In Version 8, DB2 issues SQLCODE -107 when a column name exceeds 30 bytes of UTF-8 in a CREATE TABLE, CREATE VIEW, CREATE GLOBAL TEMPORARY TABLE, ALTER TABLE ADD COLUMN, or DECLARE GLOBAL TEMPORARY TABLE statement.

Maintenance required for IBM z/OS Migration Utility

If you use the IBM z/OS Migration Utility, apply the following maintenance for IBM z/OS Migration Utility before you migrate to Version 8.

- PK26895/UK16007
- PK23287/UK14716

Ensure that you allocate enough space for complete dumps

Ensure that you allocate enough space so that you have complete dumps for problem diagnosis. Use the following MVS commands to set the MAXSPACE value to a minimum of 8000 MB:

```
DISPLAY  : D D,OPTIONS
CHNGDUMP : CD SET,SDUMP,TYPE=XMEME,MAXSPACE=4000M
```

Migrating a data sharing group

Before you migrate to compatibility mode, ensure that maintenance through the Version 8 fallback SPE is applied to all started DB2 members. If the fallback SPE is not on all active group members, Version 8 does not start but issues a message. If you have quiesced members in your data sharing group, you do not need to apply the fallback SPE to the quiesced member.

Start only one DB2 member for migration processing. During the migration, other group members can be active. However, other active group members may experience delays or timeouts if they attempt to access catalog objects that are locked by migration or enabling-new-function mode processing. After migration completes on the first member, you can migrate the other data sharing group members.

Migration of a data sharing group requires careful planning:

1. Read the information about migration considerations in this book and also in Chapter 3 of *DB2 Data Sharing: Planning and Administration*.
2. Make a plan to minimize the amount of time that some members operate at the Version 7 level and others operate at the Version 8 compatibility mode level.
3. Apply the fallback SPE to the Version 7 load library for each non-quiesced member in the data sharing group. For best availability, you can apply the SPE to one member at a time. While your data sharing group is in Version 7, you can have Version 7 subsystems with the SPE running at the same time as subsystems that are without the SPE. Stop and restart each member to activate the change.
4. Follow the procedure about migrating the data sharing group in Chapter 3 of *DB2 Data Sharing: Planning and Administration*. You must completely migrate the first member of the data sharing group to Version 8 compatibility mode before starting any other members at the Version 8 level.
5. To prepare for fallback from Version 8 compatibility mode, keep the subsystem parameter load module that is used by Version 7.
6. After all members have migrated to Version 8 compatibility mode, remain in compatibility mode until your data sharing group has processed a full range of typical work. The period of time that a data sharing group needs to remain in Version 8 compatibility mode varies depending on the size of the data sharing group and the complexity of its typical work.

The CLIST edits different jobs for enabling data sharing and migrating a data sharing member. See Chapter 3 of *DB2 Data Sharing: Planning and Administration* for the list of jobs that are edited for data sharing and migration.

Work file database size calculations

The migration job DSNTIJTC creates and updates indexes on catalog tables. These indexes are created and updated sequentially during migration. The work file database is used for the sort of each index; DB2 needs enough work file storage to sort the largest of the indexes in Table 14. Migration fails if you do not have enough storage. Therefore, ensure that you have enough space before you begin.

Table 14 shows the indexes that are new and changed for existing catalog tables.

Table 14. Indexes that are added or updated sequentially using the work file database

Catalog table name	Index name	Column names
SYSIBM.SYSCOLAUTH	SYSIBM.DSNACX01	CREATOR, TNAME, COLNAME
SYSIBM.SYSFOREIGNKEYS	SYSIBM.DSNDRH01	CREATOR, TBNAME, RELNAME
SYSIBM.SYSINDEXES	SYSIBM.DSNDXX04	INDEXTYPE
SYSIBM.SYSRELS	SYSIBM.DSNDLX02	CREATOR, TBNAME
SYSIBM.SYSSEQUENCESDEP	SYSIBM.DSNSRX02	BSCHEMA, BNAME, DTYPE
SYSIBM.SYSTABAUTH	SYSIBM.DSNATX04	TCREATOR, TNAME
SYSIBM.SYSTABLEPART	SYSIBM.DSNDPX03	DBNAME, TSNAME, LOGICAL_PART
SYSIBM.SYSTABLES	SYSIBM.DSNDTX03	TBCREATOR, TBNAME
SYSIBM.SYSVIEWDEP	SYSIBM.DSNGGX04	BCREATOR, BNAME, BTYPE, DTYPE

LOCAL DATE/TIME exits

In Version 8, if you specify a value of LOCAL for the DATE FORMAT or TIME FORMAT fields of installation panel DSNTIP4 and you do not have a replacement for each of the DB2-supplied DATE and TIME exits, DB2 might issue an error.

If you choose to use a local DATE or TIME exit, ensure that each of the three DB2-supplied exits has been replaced with your local copy of the exit.

The DB2-supplied DATE exit routine exits DSNXVDTX, DSNXVDTA, and DSNXVDTU. The DB2-supplied TIME exit routine exits DSNXVTMX, DSNXVTMA, and DSNXVTMU.

Preparing for fallback

Falling back is the process of returning DB2 to a Version 7 level after migrating your catalog and directory to Version 8 compatibility mode.

You can fall back to Version 7 only after successfully migrating the catalog to Version 8 compatibility mode by using job DSNTIJTC. However, you cannot fall back to Version 7 or return to Version 8 compatibility mode after you enter enabling-new-function or new-function mode.

Fall back if you have a severe error while operating Version 8 compatibility mode and you want to return to operation on Version 7. After fallback, the catalog remains a Version 8 catalog. If you experience a severe application or performance errors in Version 8 compatibility mode and want to return to Version 7, follow the detailed step-by-step instructions in *DB2 Installation Guide*.

To fall back to Version 7 from Version 8 compatibility mode:

1. Stop DB2 Version 8 activity.

Note: You must terminate all utilities started on Version 8.

2. Reactivate Version 7.
3. Reconnect TSO, IMS, and CICS to Version 7.
4. Start Version 7.
5. Verify fallback by running the DB2 sample applications or your own applications.

If you fall back and then try to use frozen plans or packages, the automatic rebind from the previous version fails. To make the plans and packages that were not automatically rebound on the previous version available, change the SQL statements or remove the reference to a frozen object, precompile the application programs, and explicitly BIND the plans and packages on the previous version.

Frozen objects

Falling back does not undo changes that are made to the catalog during migration to Version 8. The migrated catalog is used after fallback. Some objects in this catalog that have been affected by Version 8 function might become *frozen* objects after fallback. Frozen objects are unavailable, and they are marked with the release dependency marker L. If an object is marked with a release dependency, it is never unmarked. The release dependency marker is listed in the IBMREQD column of catalog tables.

In general, objects that depend on the new facilities of DB2 UDB for z/OS Version 8 are frozen after you fall back to Version 7 and remain frozen until you remigrate to Version 8. Table 15 lists the objects that are frozen when falling back to Version 7. Frozen objects are marked with the release dependency markers L.

Table 15. Objects that are frozen when falling back to DB2 UDB for z/OS Version 7

RELEASE DEPENDENT MARK = L
• Plans, packages, or views that use any new syntax or objects • DBRMs produced by a precompilation in Version 8 with a value of YES for the NEWFUN option • User-defined functions created in Version 8 with the PARAMETER CCSID option • User-defined SQL procedures and functions created in Version 8 with the PARAMETER CCSID option

Plans and packages become frozen objects when they use new SQL syntax, use new BIND options and attributes, or reference frozen objects. When plans and packages become frozen objects, the automatic rebind process is adversely affected. See *DB2 Installation Guide* for details.

While operating in Version 7, you can determine if any of your objects are frozen by issuing the following SELECT statements:

```
SELECT NAME FROM SYSIBM.SYSPLAN
       WHERE IBMREQD = 'L';
SELECT LOCATION, COLLID, NAME, VERSION FROM SYSIBM.SYSPACKAGE
       WHERE IBMREQD = 'L';
SELECT CREATOR, NAME FROM SYSIBM.SYSVIEWS
       WHERE IBMREQD = 'L';
SELECT CREATOR, NAME FROM SYSIBM.SYSINDEXES
       WHERE IBMREQD = 'L';
SELECT CREATOR, NAME, TYPE FROM SYSIBM.SYSTABLES
       WHERE IBMREQD = 'L';
SELECT DBNAME, NAME FROM SYSIBM.SYSTABLESPACE
       WHERE IBMREQD = 'L';
SELECT SCHEMA, NAME, SPECIFICNAME, ROUTINETYPE FROM SYSIBM.SYSROUTINES
       WHERE IBMREQD = 'L';
```

Other fallback considerations

Before you fall back to Version 7, you must be aware of the following considerations:

Buffer pools: DB2 Version 8 maintains the Version 7 virtual buffer pool and hiperpool definitions at migration so that they can be used if you fall back.

NEWFUN precompiler option: You cannot execute a plan or package that uses a DBRM that was produced by precompiling in DB2 Version 8 with a value of YES for the NEWFUN precompiler option. You cannot BIND a DBRM that was precompiled with a value of YES for the NEWFUN precompiler option on Version 7 or earlier.

DISPLAY GROUPBUFFERPOOL output: After fallback, the DISPLAY GROUPBUFFERPOOL command's output reverts to the Version 7 format and only displays the operational coupling facility level.

Utilities COPY, REPORT, and RECOVER: You must use the Version 7 COPY and RECOVER utility jobs for backup and recovery after fallback.

Running DB2-supplied stored procedure DSNWZP in a WLM-established stored procedure address space: In DB2 Version 8, DB2-supplied stored procedure DSNWZP is defined to run in a WLM-established stored procedure address space and to use external module DSNWZP. In DB2 Version 7, DSNWZP must use external module DSNWZPR to run in a WLM-established stored procedure address space. You must alter DSNWZP to use DSNWZPR after fallback.

For more information on fallback considerations, refer to *DB2 Installation Guide*.

Page-fixes for buffer pools: If you defined a buffer pool using PGFIX YES in
Version 8, it is defined with PGFIX NO after fallback. When you remigrate to
Version 8, the buffer pool is defined with PGFIX YES.

Release incompatibilities

This section describes changes that might affect your DB2 operations after migrating to Version 8 of DB2.

Ensure that Version 7 sample objects are available

If you no longer have the Version 7 sample jobs, you need to run the Version 7 installation CLIST to regenerate them. If you dropped the Version 7 sample database (by running job DSNTEJ0), you need to run the Version 7 sample jobs before you start the migration to Version 8 compatibility mode. If you do not have the Version 7 jobs available during migration, you will not have a DB2-supported sample to verify a successful migration to Version 8 compatibility mode.

Ensure that no utility jobs are running

In Version 8, you can only restart or terminate a utility on the same release on which it was started. Any outstanding utilities prior to Version 8 cannot be restarted or terminated after you have migrated from Version 7 to Version 8 compatibility mode. To ensure that you do not have outstanding utility jobs, issue the DISPLAY UTILITY(*) command.

EBCDIC and ASCII CCSID must be non-zero

You must specify a non-zero value for EBCDIC and ASCII CCSIDs. Altering of CCSIDs can be very disruptive to a system. Converting to a CCSID that supports the euro symbol is potentially less disruptive because specific pre-euro CCSIDs map to specific CCSIDs for the euro. See *DB2 Installation Guide* for the detailed steps. Converting to a different CCSID for other reasons, particularly when a DB2 subsystem has been operating with the wrong CCSID, could render data unusable and unrecoverable.

Recommendation: Never change CCSIDs on an existing DB2 subsystem without specific guidance from IBM Software Support.

Perform premigration queries (DSNTIJPM)

Job DSNTIJPM performs premigration queries to the Version 7 catalog. DSNTIJPM also generates SQL statements or utility statements to remove or correct incompatibilities. Job DSNTIJPM checks for the following incompatibilities:

- Use of data capture on DB2 catalog tables. The CATMAINT and CATENFM jobs disable data capture, which you must reactivate at a later time.
- Use of LOCKPART NO for partitioned table spaces. The LOCKPART option is deprecated in Version 8, although it is still supported for compatibility. Regardless of what you specify with the LOCKPART option, DB2 acts as though you specified LOCKPART YES, or selective partition locking (SPL).
- Partitioned table spaces that have a truncated limit key. A truncated limit key can result in an abend if the same limit key value is used for different partitions.
- Stored procedures that use LANGUAGE COMPJAVA. Stored procedures that use LANGUAGE COMPJAVA cannot be defined or run after migrating to DB2 Version 8.
- Stored procedures that use the DB2-established stored procedures address space. The DB2-established stored procedures address space is deprecated in Version 8. Existing stored procedures that use the DB2-established stored procedures address space will run in Version 8, but you cannot create or alter stored procedures using the NO WLM ENVIRONMENT clause.
- Use of the DSNWZPR module by the DB2-supplied stored procedure DSNWZP. In Version 8, DSNWZP requires a WLM stored procedure address space. Users of DSNWZPR must revert to the external module DSNWZP.
- The Version 7 sample database. To verify migration to Version 8 compatibility mode, run portions of the Version 7 sample jobs. The sample jobs require the Version 7 sample database. If the Version 7 sample database is missing, run the phase 1 and phase 2 sample jobs in Version 7 before migrating to Version 8 compatibility mode.
- Evidence of more than one code page within the same encoding scheme.
- Plans and packages for routines that need to be rebound because of a change in the DBINFO control block.
- Type 1 indexes. If you do not remove type 1 indexes from your Version 7 catalog, you will not be able to migrate to Version 8 compatibility mode.

Important: Because type 2 indexes often take more space than type 1 indexes, you should first determine if you need to allocate more space for the indexes. Refer to Section 1 of *DB2 Administration Guide* for more information. Based on these calculations, if your index space is not large enough, delete and redefine your data sets with a larger allocation. For more information about increasing your data set allocations, see *z/OS DFSMSdfp Storage Administration Reference*.

You can convert to type 2 indexes in either of the following ways:

- Use the CONVERT TO TYPE 2 option of the ALTER INDEX SQL statement. This method only works with catalog indexes, not directory indexes.
  1. Enter the SQL statement ALTER INDEX with the CONVERT TO TYPE 2 option.
  2. Run the utility REBUILD INDEX. See *DB2 Utility Guide and Reference* for more information on REBUILD INDEX.
     Recommendation: Run REBUILD INDEX on an index immediately after running ALTER INDEX because the index is unusable until it is rebuilt.

Identify unsupported objects

Version 8 does not support type 1 indexes. If you do not remove type 1 indexes from your Version 7 catalog, you will not be able to migrate to Version 8 compatibility mode.

Adjust application programs

You might need to adjust your application programs because of the release incompatibilities that this section describes.

Adjust trace applications: If you have trace applications that use statement-length fields, you might need to change them to use 4-byte statement length fields.

Adjust user-defined function calls for new built-in functions: Several new built-in functions are available. If you have user-defined functions, invoke them with a fully qualified name to avoid calling built-in functions that might have the same name. If the user-defined functions are not invoked with a fully qualified name and SYSIBM is first in the SQL path, the built-in function is selected instead of the user-defined function.

Changed defaults for utilities: The default values for several utilities options have changed in Version 8. SORTKEYS is the default for the REORG, LOAD, and REBUILD utilities. SORTDATA is the default for the REORG utility.

Changed behavior for DISPLAY LOCATION command: If you specify an empty parameter for the DISPLAY LOCATION command, such as DISPLAY LOCATION(), the command fails and DB2 issues message DSN9010I. In Version 8, you must specify a parameter for this command.

Changed output for DISPLAY GROUP command: In Version 8, the DISPLAY GROUP command displays the status of your group in compatibility mode, enabling-new-function mode, and new-function mode.

Changed behavior for TRANSLATE function: If your query references the catalog, uses the TRANSLATE built-in function, and specifies a translate table, you might need to change the translate table. In some cases, the TRANSLATE functions might not have the same behavior as in Version 7. For example, some accented characters which had a single-byte EBCDIC value in Version 7 have a double-byte Unicode value. If you perform a TRANSLATE function on a string that contains such a character, the function will not return the expected results.

Changed output for DISPLAY GROUPBUFFERPOOL command: In Version 8, the DISPLAY GROUPBUFFERPOOL command displays both the operational coupling facility level and the actual coupling facility level. In Version 7, only the operational coupling facility level is displayed.

Changed parameter length for BLOB, CLOB, and DBCLOB functions: The lower limit for these functions is now 1 for consistency with the VARCHAR and VARGRAPHIC functions. You cannot invoke these built-in functions with an explicit parameter length of 0. If you specify 0, DB2 returns an error. If the input string is empty and an explicit length is not specified, the length attribute of the result is 1.

Changed input for GRAPHIC, VARGRAPHIC, and DBCLOB: The input string for the GRAPHIC, VARGRAPHIC, and DBCLOB functions cannot be BIT data, regardless of the encoding scheme of the data. If the input string is EBCDIC BIT data, DB2 returns an error.

Changed rules for procedure and function names: In Version 8, DB2 enforces the following rules for procedure and function names:

- If your routine is written in a language other than Java, the external name is a load module which must be less than or equal to 8 bytes. The external name must contain characters that are valid for a z/OS load module.
- A procedure name cannot consist of a single asterisk.

You must change any CREATE PROCEDURE or ALTER PROCEDURE statements that specify an EXTERNAL NAME clause that does not follow the above rules if you want to execute the statements again. See *DB2 SQL Reference* for more information about these changes.

Truncation of CHAR data: Prior to Version 8, DB2 issued an error when applications invoked the CHAR function with string input data greater than 255 bytes. In Version 8, DB2 truncates the data to 255 bytes and issues a warning if non-blank characters are truncated.

SQLDA may contain truncated data: In Version 8, the length of many names has been extended. However, for compatibility with prior releases, the length of name fields in the SQLDA is not changing. Truncation of names that are stored in the SQLDA might occur with distinct name types. To avoid truncation of distinct type names in the SQLDA, you should not use distinct name types that are longer than 30 bytes. For more information, refer to *DB2 SQL Reference*.

LOCKPART has been deprecated: The LOCKPART clause on ALTER or CREATE TABLESPACE has been deprecated in Version 8, although it is still supported for compatibility purposes. In previous releases, LOCKPART determined whether individual partitions would be locked. The previous default value for LOCKPART locked the entire table space with a lock on the last partition. In Version 8 new-function mode, individual portions of partitioned tablespaces, including those created in Version 7 or earlier, will be locked as they are accessed. In a data sharing environment, all members must be in enabling-new-function or new-function mode before this change will take effect.

UTLRSTRT no longer supported: The subsystem parameter UTLRSTRT is no longer supported. When possible, DB2 attempts to restart online-restartable utilities, regardless of whether the RESTART keyword is specified.

PKGLDTOL is no longer supported: The subsystem parameter PKGLDTOL is no longer supported. DB2 Version 8 requires the package or plan for applications with the following SQL statements:

- COMMIT
- CONNECT

- DESCRIBE TABLE
- RELEASE
- ROLLBACK
- SET CONNECTION
- SET *host-variable* = CURRENT SERVER
- VALUES CURRENT SERVER INTO *host-variable*

You must bind the DBRM into a plan or package.

Restriction on DB2 private protocol applications: DB2 limits the SQL statements that a private protocol application can include to statements that were added to DB2 before Version 8.

SQL reserved words: Version 8 has several new SQL reserved words. Refer to *DB2 SQL Reference* for the list, and adjust your applications accordingly.

Changed SQL code for DATE, TIME, DATETIME, and TIMESTAMP values: In
previous releases, DB2 returned SQL code -181. In DB2 Version 8, DB2 returns SQL
code -180 in some cases. SQL codes -180 and -181 have the same SQLSTATE, 22007,
and serve a similar purpose. Adjust your applications to check for SQL codes -180
and -181 together.

Changed return code for message DSNU185: The return code for DSNU185 has changed from return code 8 to return code 0, allowing processing to continue. If you have applications that scan the return code of this message, you might need to modify them.

Input parameter markers of a prepared statement are always nullable: In previous versions of DB2, the SQLTYPE field could be set to nullable or non-nullable. In Version 8, the SQLTYPE field is always nullable.

Savepoint names cannot begin with 'SYS': A savepoint name cannot begin with SYS. If your application has a savepoint with a name beginning with SYS, DB2 will return an error.

Changed behavior for ALTER TABLESPACE: When issuing the ALTER TABLESPACE statement, partition options that are specified after an ALTER PARTITION clause will affect only the specified partition. If you do not specify an option following the specification of a partition, an error is issued.

Changed behavior for ALTER INDEX: If you do not specify an option following the ALTER PARTITION clause, a warning is issued.

EXTERNAL clause for ALTER PROCEDURE and ALTER FUNCTION requires NAME: In Version 7, the EXTERNAL clause did not require the NAME keyword followed by a value on the ALTER statement. In Version 8, you must specify NAME and a value if you specify EXTERNAL. Update your application programs to include the NAME keyword and a value when you specify the EXTERNAL clause on an ALTER statement. If you do not, DB2 issues an error message. See *DB2 SQL Reference* for more information.

Invalidation of statements that reference the catalog: In Version 8, DB2 may invalidate plans and packages that contain statements that reference the catalog. See job DSNTESQ for more information about these plans and packages.

SYSIBM.SYSDUMMY1 is recreated: During execution of job DSNTIJNE in Version 8 enabling-new-function mode processing, DB2 drops and re-creates the SYSIBM.SYSDUMMY1 table. If your plans and packages reference this table, they will be invalidated. DB2 automatically rebinds the invalidated plans and packages when the plans and packages are next references. An automatic rebind may change the access path.

Invalid uses of host variables are not supported: Validation of the attributes of host variables used in PREPARE statements is improved. You may need to update your application programs.

Example: If the defined length of the host variable is less than the length of the actual data, DB2 issues an error message. Update your application program to specify the correct length of the host variable.

Using the PARAMETER STYLE clause in CREATE PROCEDURE statement: In Version 7, you did not need to specify PARAMETER STYLE before an explicit parameter style such as SQL, DB2SQL, JAVA, GENERAL, or GENERAL WITH NULLS. In Version 8, you must specify PARAMETER STYLE ahead of any explicit parameter style, or DB2 issues an error message. If you omit the PARAMETER STYLE clause entirely, the default is PARAMETER STYLE SQL.

User IDs must have SYSOPR authority: In Version 7, DB2 commands that were issued from the z/OS console or TSO SDSF were previously associated with the SYSOPR user ID. In Version 8, these commands are associated with the primary user ID that issued them. You must grant SYSOPR authorization to these user IDs or public.

Update CCSIDs in DBINFO: If you have defined an external function or procedure with DBINFO, you might need to update the CCSIDs in DBINFO. In Version 7, CCSID fields in DBINFO were set to the CCSIDs of the invoking statement. In Version 8, a single set of three CCSIDs might not reflect the CCSIDs of a statement that invokes an external function or procedure defined by DBINFO. Adjust the CCSIDs in DBINFO, then recompile and rebind the routine that references them.

START DB2 ACCESS(MAINT) restricts access to installation SYSADM or installation SYSOPR: DB2 Version 8 enforces the restriction of START DB2 ACCESS(MAINT) that limits DB2 access to those user IDs that have installation SYSADM or installation SYSOPR authorization. Prior versions of DB2 did not enforce this restriction. For more information about the ACCESS(MAINT) option of START DB2, see *DB2 Command Reference*.

Release coexistence

This section highlights considerations for coexistence between Version 7 and Version 8 in a data sharing environment and in a distributed environment. In a data sharing environment, coexistence is limited to Version 8 compatibility mode with Version 7.

IRLM service level

As you apply IRLM service to members of a data sharing group, some members run with the newer service level, and some run with the older service level. A mix of service levels can raise issues that you must consider. For more information about IRLM coexistence, see *DB2 Data Sharing: Planning and Administration*.

DISPLAY GROUPBUFFERPOOL output

Because the DISPLAY GROUPBUFFERPOOL command output in Version 8 returns both operational and actual coupling facility levels, the command output in a coexistence environment depends on the member on which the command was issued. If the command is issued from Version 7, only the operational coupling facility level is displayed.

The coupling facility batching commands RFCOM and WARM are not used in a coexistence environment.

Distributed environment

DB2 UDB for z/OS communicates in a distributed data environment with Version 6 and Version 7 of DB2, using either DB2 private protocol access or DRDA access. However, the distributed functions that are introduced in Version 8 of DB2 UDB for z/OS can be used only when using DRDA access.

Other DRDA partners at DRDA level 4 can also take advantage of the functions that are introduced in Version 8 of DB2 UDB for z/OS.

Data sharing

DB2 can support both Version 7 and Version 8 members in compatibility mode in a data sharing group. To support both releases, you must first apply the fallback SPE to all Version 7 members of the group. Release coexistence begins when you migrate the first data sharing member to Version 8. You must successfully migrate the first data sharing member to Version 8 before attempting to migrate the other data sharing members.

For the best availability, you can migrate the members to Version 8 one member at a time. When developing your migration plan, remember that most new functions that are introduced in Version 8 are not available to any members of the group until all members are migrated to Version 8 and until all members are in new-function mode.

For detailed information about data sharing release coexistence considerations, see *DB2 Data Sharing: Planning and Administration*.

TSO and CAF logon procedures: You can attach to either release of DB2 with your existing TSO or CAF logon procedures, without changing the load libraries for your applications. After you migrate completely to the latest level of DB2, you must update those procedures and jobs to point to the latest level of DB2 load libraries. If you forget to update those procedures and jobs before migrating to any release subsequent to Version 8, those procedures and jobs can no longer work in that subsequent release.

For a detailed list of considerations for a data sharing group with multiple DB2 releases, see Chapter 3 of *DB2 Data Sharing: Planning and Administration*.

Installation changes

This section shows the panels that are used by the installation CLIST to customize the jobs that you use to install or migrate to Version 8. This section also lists the changes to SMP/E jobs and sample jobs.

You can also install DB2 UDB for z/OS from a Windows workstation using mSys for Setup DB2 Customization Center.

Version 8 panels

Table 16 lists the panels for DB2 UDB for z/OS installation and migration. With the addition of the new functions in Version 8, several panels have been modified, and new fields have been added. The new and modified panels have a Yes listed under the **Panel modified** column in Table 16.

Table 16. Version 8 installation and migration panels

Panel ID	Panel title	Panel modified
DSNTIPA0	Online Book Data Set Names	
DSNTIPA1	Main Panel	Yes
DSNTIPA2	Data Parameters	
DSNTIPK(1)	Define Group or Member	
DSNTIPH	System Resource Data Set Names	
DSNTIPT	Data Set Names Panel 1	
DSNTIPU	Data Set Names Panel 2	Yes
DSNTIPW	Data Set Names Panel 3	Yes
DSNTIPD	Sizes Panel 1	
DSNTIP7	Sizes Panel 2	Yes
DSNTIPE	Thread Management	Yes
DSNTIP1	Buffer Pool Sizes Panel 1	Yes
DSNTIP2	Buffer Pool Sizes Panel 2	Yes
DSNTIPN	Tracing and Checkpoint Parameters	Yes
DSNTIPO	Operator Functions	
DSNTIPF	Application Programming Defaults Panel 1	Yes
DSNTIP4	Application Programming Defaults Panel 2	Yes
DSNTIP8	Application Programming Defaults Panel 3	Yes
DSNTIPI	IRLM Panel 1	Yes
DSNTIPJ	IRLM Panel 2	Yes
DSNTIPP	Protection	Yes
DSNTIPM	MVS PARMLIB Updates	
DSNTIPL	Active Log Data Set Parameters	Yes
DSNTIPA	Archive Log Data Set Parameters	Yes
DSNTIPS	Databases and Spaces to Start Automatically	
DSNTIPR	Distributed Data Facility Panel 1	
DSNTIP5	Distributed Data Facility Panel 2	
DSNTIPX	Routine Parameters	Yes
DSNTIPZ	Data Definition Control Support	
DSNTIPY	Job Editing	Yes
DSNTIPC	DB2 CLIST Calculations Panel 1	Yes
DSNTIPC1	DB2 CLIST Calculations Panel 2	

Table 16. Version 8 installation and migration panels (continued)

Panel ID	Panel title	Panel modified
DSNTIPB	Update Selection Menu	Yes

Notes:

1. DSNTIPK is for installing and migrating in data sharing mode.

Version 8 sample jobs

With the addition of the new functions in Version 8, several existing sample jobs have been modified, and several new jobs have been added. The new and changed sample jobs are listed in Table 17.

Table 17. New and modified sample jobs

Sample job	New or modified
DSNTEJ3M	New
DSNTEJ6R	New
DSNTEJ76	New
DSNTEJ77	New
DSNTEJ78	New
DSNTEJ65	Modified
DSNTEJ1	Modified
DSNTEJ1P	Modified
DSNTEJ2A	Modified

Appendix A. Changes to commands

This appendix provides an overview of the new and changed commands in Version 8 of DB2 UDB for z/OS. The purpose of the appendix is to highlight the major changes. The following topics provide additional information:

- "New commands"
- "Changed commands"
- "Other command changes" on page 93

For complete information about all the changes, such as the syntax for new or changed commands, see *DB2 Command Reference*.

New commands

Table 18 shows the new commands in Version 8.

Table 18. New commands

Command name	Description
MODIFY admtproc,APPL=SHUTDOWN	Stops the administrative scheduler from accepting requests and executing new tasks, and shuts down the administrative scheduler.
MODIFY admtproc,APPL=TRACE	Starts or stops traces in the administrative scheduler.
START admtproc	Starts the administrative scheduler that is specified in the *admtproc* parameter.
STOP admtproc	Stops the administrative scheduler that is specified in the *admproc* parameter.

Changed commands

Table 19 shows that several existing commands have new and changed options.

Table 19. Changes to existing commands

Command	Description of enhancements and notes
-ALTER BUFFERPOOL (DB2)	**New option:** PGFIX(NO | YES) The PGFIX option specifies whether the buffer pool should be fixed in real storage when it is used.

Table 19. Changes to existing commands (continued)

Command	Description of enhancements and notes
BIND PLAN (DSN) BIND PACKAGE (DSN) REBIND PLAN (DSN) REBIND PACKAGE (DSN)	**New and changed options:** REOPT(NONE) REOPT(ALWAYS) REOPT(ONCE) The REOPT option specifies whether to have DB2 determine an access path at run time by using the values of host variables, parameter markers, and special registers. REOPT(NONE) does not determine an access path at run time. You can use NOREOPT(VARS) as a synonym for REOPT(NONE). REOPT(ALWAYS) determines the access path at run time each time the statement is run. You can use REOPT(VARS) as a synonym for REOPT(ALWAYS). REOPT(ONCE) determines the access path for any dynamic statement only once, at the first run time or at the first time the statement is opened. This access path is saved in the dynamic statement cache and used until the statement is invalidated or removed from the cache and needs to be prepared again.
-DISPLAY DATABASE (DB2)	**New options:** OVERVIEW ADVISORY(ARBDP) ADVISORY(AREO*) OVERVIEW displays each object in the database on its own line, providing an easy way to see all objects in the database. ADVISORY(ARBDP) displays objects that are in the advisory REBUILD-pending status. ADVISORY(AREO*) displays objects that are in the advisory REORG-pending status. In Version 8, you can use the DISPLAY DATABASE command on the following objects: • Databases • Table spaces • Index spaces • Physical partitions of partitioned table spaces or index spaces (including index spaces that contain data-partitioned secondary indexes) • Logical partitions of nonpartitioned secondary indexes
-DISPLAY GROUP (DB2)	The DISPLAY GROUP command with DETAIL option now displays the catalog mode in the output as MODE(C|E|N) (compatibility mode, enabling-new-function mode, or Version 8 new-function mode) of the DB2 subsystem or data sharing group.

Table 19. Changes to existing commands (continued)

Command	Description of enhancements and notes
MODIFY irlmproc,SET (z/OS IRLM)	**New and changed options:** DEADLOCK=*nnnn* PVT=*nnnn* DEADLOCK specifies the number, in milliseconds, of how often the local deadlock processing is scheduled. PVT specifies the upper limit of private storage that is used for locks. You can specify this value in megabytes or gigabytes by specifying M (for megabytes) or G (for gigabytes) after the value, as follows, *nnnn*M or *nnnn*G.
-START DATABASE (DB2)	In Version 8, you can use the START DATABASE command on the following objects: • Databases • Table spaces • Index spaces • Physical partitions of partitioned table spaces or index spaces (including index spaces that contain data-partitioned secondary indexes) • Logical partitions of nonpartitioned secondary indexes
START irlmproc (z/OS IRLM)	**New and changed options:** LTE=*nnnn* MAXCSA= PC= LTE specifies the number of lock table entries that are required in the coupling facility lock structure. MAXCSA is a required positional parameter but is currently unused. PC is a required positional parameter but is currently unused. MAXCSA and PC are currently unused because IRLM Version 2 Release 2 places locks only in private storage.
-STOP DATABASE (DB2)	In Version 8, you can use the STOP DATABASE command on the following objects: • Databases • Table spaces • Index spaces • Physical partitions of partitioned table spaces or index spaces (including index spaces that contain data–partitioned secondary indexes) • Logical partitions of nonpartitioned secondary indexes

Other command changes

If secondary authorization IDs are defined, DB2 commands that are issued from a z/OS console or TSO SDSF are associated with those IDs.

Appendix B. Changes to utilities

This appendix summarizes the changes to utilities in Version 8 of DB2 UDB for z/OS. The following topics provide additional information:

- "New utilities"
- "Changed utilities"
- "Other utility changes" on page 102

New utilities

Table 20 lists and describes the new utilities.

Table 20. Overview of new utilities

Utility name	Description
BACKUP SYSTEM	BACKUP SYSTEM takes fast volume-level copies of DB2 databases and logs. It relies on new DFSMShsm services in z/OS Version 1 Release 5 that automatically monitors which volumes need to be copied. Using BACKUP SYSTEM to take copies is less disruptive than using the SET LOG SUSPEND command, because a BACKUP SYSTEM job does not take a log write latch. An advantage for data sharing is that BACKUP SYSTEM operates on an entire data-sharing group, whereas the SET LOG SUSPEND command must be issued for each data-sharing member.
CATENFM	CATENFM enables a DB2 subsystem to enter DB2 Version 8 enabling-new-function mode and Version 8 new-function mode.
RESTORE SYSTEM	RESTORE SYSTEM provides a way to recover a DB2 subsystem to an arbitrary point in time. RESTORE SYSTEM automatically handles any creates, drops, and LOG NO events that might have occurred between the backup and the recovery point in time. RESTORE SYSTEM uses data that is copied by the BACKUP SYSTEM utility.
DSNJCNVB	The DSNJCNVB stand-alone conversion utility converts the bootstrap data set (BSDS) so that it can support up to 10 000 archive log volumes and 93 active log data sets per log copy. If you do not convert the BSDS, it can manage only 1000 archive log volumes and 31 active log data sets per log copy.

Changed utilities

Table 21 on page 96 lists and describes the new and changed options for many existing DB2 UDB for z/OS utilities.

Table 21. New and changed utility options

Utility name	Description of enhancements and notes
CHECK INDEX	**New options:** SHRLEVEL CHANGE, SHRLEVEL REFERENCE, DRAIN_WAIT, RETRY, RETRY_DELAY SHRLEVEL CHANGE enables CHECK INDEX to operate online. When you specify this option, applications can read from and write to data that is to be checked. To prevent applications from writing to the data that is to be checked, specify SHRLEVEL REFERENCE. The DRAIN_WAIT, RETRY, and RETRY_DELAY options govern the behavior of the utility when draining the table space or index. The DRAIN_WAIT option specifies the number of seconds that CHECK INDEX is to wait. The RETRY option specifies the number of retries that CHECK INDEX is to attempt. The RETRY_DELAY option specifies the minimum duration, in seconds, between retries.
CHECK LOB	**Changed option:** WORKDDN The WORKDDN keyword, which provided the DD names of the SYSUT1 and SORTOUT data sets in earlier versions of DB2, is no longer needed and is ignored. You do not need to modify existing control statements to remove WORKDDN.
COPY	**New option:** SYSTEMPAGES The SYSTEMPAGES option specifies whether the COPY utility is to put the system pages at the beginning of the image copy data set. SYSTEMPAGES YES, which is the default, guarantees that the system pages are located at the beginning of the image copy. This placement ensures that the image copy contains the necessary system pages for subsequent UNLOAD utility jobs to correctly format and unload all data rows. You can now specify COPY CONCURRENT SHRLEVEL CHANGE for table spaces with a page size that is greater than 4 KB if the page size matches the control interval for the associated data set. (DB2 now supports data sets with control interval sizes of 8 KB, 16 KB, and 32 KB.)

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
LOAD	**New options:** BIT, BLOBF, DELIMITED, COLDEL, CHARDEL, CLOBF, DBCLOBF, DECPT, STRIP, TRUNCATE **Changed options:** HISTORY, UPDATE, SORTKEYS The BIT option specifies that the input CHAR or VARCHAR field contains BIT data. You can load delimited files by specifying the DELIMITED option. (This support for delimited files improves compatibility with the DB2 family.) You can use the COLDEL, CHARDEL, and DECPT options to specify the delimiter characters that are used in the input data file. These options have the following meanings: **COLDEL** Specifies the column delimiter character. **CHARDEL** Specifies the character string delimiter character. **DECPT** Specifies the decimal point delimiter character. The STRIP option specifies that LOAD is to remove blanks (the default) or the specified character from the beginning, end, or both ends of the input data. The TRUNCATE option specifies that LOAD is to truncate the input character string from the right if the string does not fit in the target column. These options are valid only with the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data type options. If you specify both the TRUNCATE and STRIP options, LOAD performs the strip operation first. The UPDATE and HISTORY options are now independent of each other. The value that you specify for UPDATE does not determine the value that you can specify for HISTORY. The SORTKEYS option is now forced on for LOAD. DB2 provides a method for loading LOB data that is over 32 KB in length. The field for a LOB in the input data set can contain a file name. This file name is for the file that contains the LOB data. The data type for a LOB that is input in this way is CHAR BLOBF, VARCHAR BLOBF, CHAR CLOBF, VARCHAR CLOBF, CHAR DBCLOBF, or VARCHAR DBCLOBF.

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
REBUILD INDEX	**New options:** INDEXSPACE, SCOPE **Changed options:** HISTORY, UPDATE, SORTKEYS The INDEXSPACE option allows you to identify the index by specifying the qualified name of the index space, which you can obtain from the SYSIBM.SYSINDEXES table. The SCOPE option indicates whether to rebuild all specified indexes (SCOPE ALL) or to rebuild only those indexes that are in REBUILD-pending, RECOVER-pending, advisory REBUILD-pending, or advisory REORG-pending status (SCOPE PENDING). The UPDATE and HISTORY options are now independent of each other. The value that you specify for UPDATE does not determine the value that you can specify for HISTORY. The SORTKEYS option is now forced on for REBUILD INDEX.
RECOVER	**New option:** CURRENTCOPYONLY CURRENTCOPYONLY specifies that the restore process is to use only the most recent primary copy for each object in the list. If restore fails, RECOVER does not automatically use the next most recent copy or the backup copy, and the object fails. Specify CURRENTCOPYONLY to improve the performance of restoring concurrent copies (copies that were made by COPY with the CONCURRENT option). Certain catalog and directory objects can be grouped together for recovery. You can specify them as a list of objects in a single RECOVER utility statement. When you specify all of these objects in one statement, these objects are recovered faster.
REORG INDEX	**New options:** INDEXSPACE, SORTDEVT, SORTNUM **Changed options:** HISTORY, UPDATE The INDEXSPACE option allows you to identify the index by specifying the qualified name of the index space, which you can obtain from the SYSIBM.SYSINDEXES table. Use the SORTDEVT option to specify the device type for temporary data sets that are to be allocated by DFSORT™. Use the SORTNUM option to specify the number of these data sets. The UPDATE and HISTORY options are now independent of each other. The value that you specify for UPDATE does not determine the value that you can specify for HISTORY.

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
REORG TABLESPACE	**New options:** SCOPE, REBALANCE **Changed options:** HISTORY, UPDATE, SORTDATA, SORTKEYS, DISCARD The SCOPE option indicates whether to reorganize the specified table space or partitions (SCOPE ALL) or to reorganize the specified table space or partitions only if they are in REORG-pending or advisory REORG-pending status (SCOPE PENDING). REBALANCE specifies that REORG TABLESPACE is to set new partition boundaries so that all rows that participate in the reorganization are evenly distributed across the reorganized partitions. The UPDATE and HISTORY options are now independent of each other. The value that you specify for UPDATE does not determine the value that you can specify for HISTORY. The SORTDATA and SORTKEYS options are forced on for REORG TABLESPACE. You can now specify the DISCARD option with SHRLEVEL CHANGE. However, if you specify these two options together, data rows that match the discard criteria cannot be modified during reorganization. You can reorganize catalog table spaces with links if you specify SHRELEVEL REFERENCE.
REPAIR	**New options:** VERSIONS, INDEXSPACE, NOAREOPENDSTAR # **Changed options:** # DBD REBUILD The VERSIONS option updates the version information for the named table space or index in the catalog and directory. Use this option when you are moving objects from one system to another or as a part of version number management. The INDEXSPACE option allows you to identify the index by specifying the qualified name of the index space, which you can obtain from the SYSIBM.SYSINDEXES table. You can use the NOAREOPENDSTAR option to reset the advisory REORG-pending (AREO*) status of the specified table space or index. # You no longer need to start a database for access by utilities only before REPAIR # DBD REBUILD could be performed. DB2 now performs this step for you.

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
RUNSTATS	**New options:** COLGROUP, MOST, LEAST, BOTH, SORTDEVT, SORTNUM **Changed options:** FREQVAL, COUNT, HISTORY, UPDATE, INDEX LIST The COLGROUP option indicates that RUNSTATS should treat the specified columns as a group. This option allows RUNSTATS to collect a cardinality value on the group. The FREQVAL and COUNT options can be specified with COLGROUP to indicate that RUNSTATS is to collect frequency statistics for the specified group of columns. If you specify FREQVAL, you must specify COUNT and COLGROUP. The COUNT option indicates the number of frequently occurring values that are to be collected. You must specify a value; no default exists. The MOST, LEAST, and BOTH options indicate whether RUNSTATS is to collect the most frequently occurring values for the set of columns (MOST), the least frequently occurring values for the set of columns (LEAST), or both (BOTH). These options must be specified with the FREQVAL and COUNT options. The SORTDEVT option specifies the device type for temporary data sets that are to be allocated by DFSORT. The SORTNUM option specifies the number of these data sets. The UPDATE and HISTORY options are now independent of each other. The value that you specify for UPDATE does not determine the value that you can specify for HISTORY. Support for the correlation stats-spec keywords is added, when specified along with the RUNSTATS INDEX LIST keywords.
TEMPLATE	**New options:** DIR, DSNTYPE The following items are added to support the unloading of LOB data that is greater than 32KB: • DIR specifies the number of directory blocks for a partitioned data set that will hold the LOB data. • DSNTYPE specifies the type of data set that is to be allocated for the LOB data. • The &UNIQ. variable specifies a unique suffix that is appended to dynamically allocated output data set names for UNLOAD.

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
UNLOAD	**New options:** BLOBF, COLDEL, CHARDEL, CLOBF, DBCLOBF, DECPT, DELIMITED, FROMSEQNO You can unload files in delimited format by specifying the DELIMITED option. (This support for delimited files improves compatibility with the DB2 family.) You can use the COLDEL, CHARDEL, and DECPT options to specify the delimiter characters that are to be used. These options have the following meanings: **COLDEL** Specifies the column delimiter character. **CHARDEL** Specifies the character string delimiter character. **DECPT** Specifies the decimal point delimiter character. The FROMSEQNO option specifies the file sequence number of the image copy data set from which data is to be unloaded. This option enables you to unload data from tape data sets that are not cataloged. DB2 provides a method for unloading LOB data that is over 32KB in length. If the data type for an output field is CHAR BLOBF, VARCHAR BLOBF, CHAR CLOBF, VARCHAR CLOBF, CHAR DBCLOBF, or VARCHAR DBCLOBF, that field in the output data set contains a file name. This file name is for the file that contains the LOB data.
DSNJU003 (change log inventory)	**New options:** ALIAS=*alias-name*:*alias-port*, NOALIAS, SYSPITR=*log-truncation-point*, CCSIDS **Changed option:** ENDLRSN The ALIAS option (on the DDF statement) specifies one to eight alias names for the location. *:alias-port* specifies a TCP/IP port number for the alias that can be used by DDF to accept distributed requests. NOALIAS indicates that no alias names exist for the specified location. Any alias names that were specified in a previous DSNJU003 utility job are removed. Before you run RESTORE SYSTEM to recover system data, you must use the SYSPITR option of DSNJU003. The SYSPITR option specifies the log RBA (non-data sharing system) or the log LRSN (data sharing system) that represents the log truncation point for the point in time that you want to use for system recovery. The CCSIDS option can be specified on the DELETE statement to delete the CCSID information in the BSDS. CCSID information is stored in the BSDS to ensure that you do not accidentally change the CCSID values. Use this option under the direction of IBM Software Support when the CCSID information in the BSDS is incorrect. After you run a DSNJU003 job with the DELETE CCSIDS option, the CCISD values from DSNHDECP are recorded in the BSDS the next time DB2 is started. The ENDLRSN option can now be used with CRESTART in a non-data sharing environment. In this environment, ENDLRSN specifies the RBA value that matches the start of the last log record that is to be used during restart. Any log information in the bootstrap data set, the active logs, and the archive logs with an RBA that is greater than the ENDLRSN value is discarded. If the ENDLRSN RBA value does not match the start of a log record, DB2 restart fails.

Table 21. New and changed utility options (continued)

Utility name	Description of enhancements and notes
DSN1COPY	**New options:** EBCDIC, ASCII, UNICODE You can indicate the format of the row data in the PRINT output by specifying EBCDIC, ASCII, or UNICODE with the PRINT option.
DSN1PRNT	**New options:** EBCDIC, ASCII, UNICODE You can indicate the format of the row data in the PRINT output by specifying EBCDIC, ASCII, or UNICODE with the PRINT option.

Other utility changes

Other changes to utilities in Version 8 are:

- You can reset the advisory REORG-pending (AREO*) status by running the REBUILD INDEX utility, the REORG INDEX utility, the REORG TABLESPACE utility, the REPAIR utility, or the LOAD utility with the REPLACE option.
- You can reset the advisory REBUILD-pending (ARBDP) status by running the REBUILD INDEX utility, the REORG TABLESPACE utility, the REPAIR utility, or the LOAD utility with the REPLACE option.
- You can code your utility control statements either entirely in EBCDIC or entirely in Unicode UTF-8; do not mix character sets. DB2 automatically detects and processes Unicode UTF-8 control statements if the first character of the data set is one of the following characters:
  - A Unicode UTF-8 blank (X'20')
  - A Unicode UTF-8 dash (X'2D')
  - A Unicode UTF-8 uppercase characters A through Z (X'41' through X'5A')
- The new stored procedure DSNUTILU lets you invoke utilities from a local or remote client program that generates Unicode utility control statements.
- You can restart utility jobs without specifying the RESTART keyword. If you resubmit a job that finished abnormally, DB2 recognizes the job and restarts it if possible.

Appendix C. Changes to SQL

This appendix provides an overview of the new and changed SQL statements in Version 8 of DB2 UDB for z/OS. The following topics provide additional information:

- "New SQL statements"
- "Changed SQL statements"
- "New functions" on page 117
- "Other SQL language changes" on page 118

For complete information about all changes, such as the syntax for new or changed SQL statements, comprehensive descriptions of keywords, and examples of usage, see *DB2 SQL Reference*.

New SQL statements

Table 22 shows the new SQL statements in Version 8.

Table 22. New SQL statements

SQL statement	Description
ALTER SEQUENCE	Changes the description of a sequence object
ALTER VIEW	Regenerates a view using an existing view definition at the current server
CREATE SEQUENCE	Defines a sequence object
GRANT (sequence privileges)	Grants privileges on a user-defined sequence object
ITERATE	Causes the flow of control within an SQL procedure to return to the beginning of a labelled loop
REFRESH TABLE	Refreshes the data in a materialized query table
RESIGNAL	Enables a condition handler within an SQL procedure to raise a condition with a specific SQLSTATE and message text, or to return the same condition that activated the handler
RETURN	Returns status information from an SQL procedure
REVOKE (sequence privileges)	Revokes privileges on a user-defined sequence object
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	Assigns a value to the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register
SET CURRENT PACKAGE PATH	Assigns a value to the CURRENT PACKAGE PATH special register
SET CURRENT REFRESH AGE	Assigns a value to the CURRENT REFRESH AGE special register
SET ENCRYPTION PASSWORD	Assigns a value for the ENCRYPTION PASSWORD and, optionally, a hint for the password
SET SCHEMA	Assigns a value to the CURRENT SCHEMA special register.

Changed SQL statements

As shown in Table 23 on page 104, many existing SQL statements have new and changed clauses.

Table 23. Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ALTER FUNCTION (external)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE **Deprecated clauses:** LANGUAGE COMPJAVA The new clauses let you specify whether the function is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. You can no longer specify LANGUAGE COMPJAVA because support for LANGUAGE COMPJAVA stored procedures is removed.
ALTER INDEX	**New clauses:** PADDED NOT PADDED CLUSTER NOT CLUSTER ADD COLUMN ALTER PARTITION ENDING AT # MAXVALUE When an index contains at least one varying-length column, PADDED and NOT PADDED specify how the varying-length columns in the index are to be stored. PADDED indicates that the varying-length string columns in the index are padded with the default pad character to their maximum length. NOT PADDED indicates that the varying-length string columns are not padded to their maximum length. The partitioning index for a table is no longer required to be the clustering index. You can use the new CLUSTER and NOT CLUSTER clauses to change which index is the clustering index for a table. The new ADD COLUMN clause lets you add a column to an existing index. In previous releases of DB2, to change the attributes of a partition of a partitioning index, you used the PART and VALUES keywords. Although these keywords are still supported in Version 8, the ALTER PARTITION and ENDING AT keywords are the # preferred syntax to use when changing a partition's attributes. The MAXVALUE # keyword can be specified in conjunction with the ENDING AT keyword to specify # that the partition identified by the ALTER PARTITION keyword is changed to end at # the maximum value.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
ALTER PROCEDURE (external)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE **Deprecated clauses:** LANGUAGE COMPJAVA NO WLM ENVIRONMENT The new clauses let you specify whether the procedure is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. You can no longer specify LANGUAGE COMPJAVA because support for LANGUAGE COMPJAVA stored procedures is removed. You can no longer specify NO WLM ENVIRONMENT. Stored procedures that are created or altered in Version 8 must run in a WLM-established address space.
ALTER PROCEDURE (SQL)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE **Deprecated clauses:** NO WLM ENVIRONMENT The new clauses let you specify whether the procedure is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. You can no longer specify NO WLM ENVIRONMENT. Stored procedures that are created or altered in Version 8 must run in a WLM-established address space.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
ALTER TABLE	**New clauses:** ADD PARTITION BY ADD PARTITION ENDING AT ALTER PARTITION ENDING AT ROTATE PARTITION FIRST TO LAST ADD MATERIALIZED QUERY ALTER MATERIALIZED QUERY DROP MATERIALIZED QUERY VOLATILE NOT VOLATILE AS SECURITY LABEL MAXVALUE **Changed clauses:** ALTER *column-alteration* *referential-constraint* AS IDENTITY GENERATED ALWAYS GENERATED BY DEFAULT In Version 8 of DB2, the data partitions for a partitioned index can be determined either by the definition of a partitioning index (index-controlled partitioning) or by the definition of table itself (table-controlled partitioning). In support of table-controlled partitioning, the ALTER TABLE statement has several new clauses: • When the partitioning for a table is yet to be established (it has neither index- nor table-controlled partitioning), you can use the ADD PARTITION BY clause to make the table partitioning be table-controlled. The clause specifies the columns that are used to partition the data, the number of partitions, and the limit keys for the partition boundaries. • For an existing partitioned table (whether it has index- or table-controlled partitioning), you can use the ADD PARTITION ENDING AT clause to define a new partition for the table, the ALTER PARTITION ENDING AT clause to change the limit keys for the partitions of the table, and use the ROTATE PARTITION FIRST TO LAST clause to rotate the partitions such that the first logical partition becomes the last logical partition. You can also specify the MAXVALUE keyword with each of these clauses to change the limit key for the specified partition to the maximum value. The ALTER TABLE statement has several new clauses to provide support for materialized query tables: • Use ADD MATERIALIZED QUERY to change an existing base table into a materialized query table. This clause defines the fullselect on which the materialized query table is based. • Use ALTER MATERIALIZED QUERY to modify the attributes of an existing materialized query table. This clause supports changing the attributes that are defined by MAINTAINED BY SYSTEM or MAINTAINED BY USER and ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION. • Use ALTER MATERIALIZED QUERY to change a materialized query table into a base table. The clause causes DB2 to stop treating the table as a materialized query table. The new VOLATILE and NOT VOLATILE clauses control how DB2 tries to access the table for SQL operations. VOLATILE specifies that DB2 should use index access for the table whenever possible. When you add a column to a table, you can specify the AS SECURITY LABEL clause. A security label column indicates that the table is defined with multilevel security with row level granularity. The security label column contains the security label values.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
ALTER TABLE (cont)	In previous versions of DB2, the changes that you could make to an existing column definition were limited. You could increase only the length of VARCHAR columns. In Version 8, you can change the data type of many columns, such as from SMALLINT to INTEGER or VARCHAR to CHAR. You can also now change the attributes of an existing identity column. When you add an identity column to a table, you can specify new keywords on the AS IDENTITY clause. ORDER and NO ORDER indicate whether the identity values must be generated in order of request. The default is NO ORDER. You can also use the new keywords NO MAXVALUE and NO MINVALUE to explicitly specify the default behavior that an identity column has no maximum or minimum value unless you specify one. In addition, you can now specify 0 as the interval between sequence values and set the minimum and maximum values to the same value, which effectively lets you define a constant sequence such that the same value is always returned. Many new clauses are also added to the ALTER TABLE statement that let you change any of the attributes of an existing identity column. When you define a referential constraint, the new keywords ENFORCED or NOT ENFORCED indicate whether DB2 enforces the constraint or treats the constraint as an informational referential constraint. DB2 assumes that the user enforces the constraint when it is defined as NOT ENFORCED, or informational. When you add a ROWID column to a table, you are no longer required have to explicitly specify GENERATED ALWAYS or GENERATED BY DEFAULT. The default is GENERATED ALWAYS. A ROWID column does not need to exist when you add a LOB column to a table. If a ROWID column does not exist, DB2 implicitly generates one and appends it as the last column of the table. An implicitly generated ROWID column is called a *hidden ROWID column*.
ALTER TABLESPACE	**New clauses:** ALTER PARTITION **Deprecated clauses:** LOCKPART When modifying the attributes of a partition, you can specify ALTER PARTITION instead of PART to identify the partition to alter. Although the PART keyword is still supported as a synonym, ALTER PARTITION is the new preferred syntax. In addition, any options specified after the ALTER PARTITION clause affect only the specified partition; if no options are specified for the specified partition, an error occurs. In Version 8, DB2 treats all partitioned table spaces as if they were defined with LOCKPART YES. You can still specify the LOCKPART clause, but it has no effect. When all the conditions for selective partition locking are met, DB2 locks only the partitions that are accessed. When the conditions for selective partition locking are not met, DB2 locks every partition of the table space.
COMMENT	**New clauses:** PACKAGE PLAN SEQUENCE If you have the appropriate authorization, you can use the new PACKAGE, PLAN, and SEQUENCE clauses to provide comments for packages, plans, and sequences in the DB2 catalog. When creating a comment for a package, you can specify which version of the package to which the comment applies.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
CREATE FUNCTION (external scalar)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE The new clauses let you specify whether the function is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. The default is STOP AFTER SYSTEM DEFAULT FAILURES.
CREATE FUNCTION (external table)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE The new clauses let you specify whether the function is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT field (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. The default is STOP AFTER SYSTEM DEFAULT FAILURES.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
CREATE INDEX	**New clauses:** PADDED NOT PADDED PARTITIONED NOT CLUSTER PARTITION BY RANGE # MAXVALUE **Changed clauses:** CLUSTER When an index contains at least one varying-length column, PADDED and NOT PADDED specify how the varying-length columns in the index are to be stored. PADDED indicates that the varying-length string columns in the index are padded with the default pad character to their maximum length. NOT PADDED indicates that the varying-length string columns are not padded to their maximum length. The default for the option is determined by the value of field PAD INDEXES BY DEFAULT (on installation panel DSNTIPE). For new installations of DB2, the default is for PADDED mode. For migrations to Version 8, the default is NOT PADDED mode. When you create an index for a partitioned table, you can define the index as PARTITIONED. PARTITIONED indicates that the index is data partitioned (that is, the index is partitioned according to the partitioning scheme of the underlying data). Both partitioning indexes, when the partitioning of tables is index-controlled, and secondary indexes can be data partitioned. The partitioning index for a table is no longer required to be the clustering index. You can use CLUSTER and NOT CLUSTER to specify which index is the clustering index. The default is NOT CLUSTER. In previous releases of DB2, you had to use the PART keyword to define a partitioning index. In Version 8, if you are using index-controlled partitioning and are defining a partitioning index, you can use the new PARTITION BY RANGE and PARTITION keywords to specify the partitions. The PART keyword is supported for # compatibility, but the new syntax is preferred for clarity. You can specify the # ENDING AT MAXVALUE clause with the PARTITION keyword to set the limit key # for the specified partition to the maximum value.
CREATE PROCEDURE (external)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE **Deprecated clauses:** LANGUAGE COMPJAVA NO WLM ENVIRONMENT The new clauses let you specify whether the procedure is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to placed in a stopped state after any failure. The default is STOP AFTER SYSTEM DEFAULT FAILURES. You can no longer specify LANGUAGE COMPJAVA because support for LANGUAGE COMPJAVA stored procedures is removed. You can no longer specify NO WLM ENVIRONMENT. Stored procedures that are created or altered in Version 8 must run in a WLM-established address space.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
CREATE PROCEDURE (SQL)	**New clauses:** STOP AFTER *nn* FAILURES STOP AFTER SYSTEM DEFAULT FAILURES CONTINUE AFTER FAILURE **Changed clauses:** NO WLM ENVIRONMENT The new clauses let you specify whether the procedure is to be placed in a stopped state after some number of failures. You can specify a specific number, specify that DB2 is to use the value of field MAX ABEND COUNT (on installation panel DSNTIPX), or specify that the routine is not to be placed in a stopped state after any failure. The default is STOP AFTER SYSTEM DEFAULT FAILURES. You can no longer specify NO WLM ENVIRONMENT. Stored procedures that are created or altered in Version 8 must run in a WLM-established address space. In addition to changes to the CREATE PROCEDURE (SQL) statement itself, you can specify several new or enhanced statements in an SQL procedure. These statements let the procedure return more information to the caller of the procedure. The new RETURN statement supports returning status information. GET DIAGNOSTICS is extended to support returning the status information from a RETURN statement. You can also use SIGNAL and RESIGNAL to return a condition with a specific SQLSTATE and message text.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
CREATE TABLE	**New clauses:** *partitioning-clause* *materialized-query-definition* EXCLUDING IDENTITY COLUMN DEFAULTS VOLATILE NOT VOLATILE AS SECURITY LABEL MAXVALUE **Changed clauses:** *referential-constraint* AS IDENTITY GENERATED ALWAYS GENERATED BY DEFAULT The *partitioning-clause* lets you define a partitioned table with table-controlled partitioning. The PARTITION BY keyword specifies the columns that are used to partition the data. The PARTITION and ENDING AT keywords specify the number of partitions and the limit keys for the partition boundaries. The MAXVALUE keyword, when specified with the ENDING AT keyword, specifies that the limit key value for a partition is set to the maximum value. The *materialized-query-definition* lets you specify a fullselect to define the columns of the table and to indicate whether the table is to be a materialized query table. If the table is not to be used as a materialized query table, you must specify WITH NO DATA, and you can use keywords to specify how the identity column and column default attributes are to be inherited. The following clauses are used to define the attributes of a table that is to be used as a materialized query table: • DATA INITIALLY DEFERRED • REFRESH DEFERRED • MAINTAINED BY SYSTEM or MAINTAINED BY USER • ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION In previous releases of DB2, INCLUDING IDENTITY COLUMN DEFAULTS was introduced as an optional clause to be used with the LIKE clause. A clause did not exist for the default behavior. You can now explicitly specify EXCLUDING IDENTITY COLUMN DEFAULT to indicate the default behavior. The new VOLATILE and NOT VOLATILE keywords control how DB2 tries to access the table for SQL operations. VOLATILE specifies that DB2 should use index access for the table whenever possible. The default is NOT VOLATILE. You can specify the AS SECURITY LABEL clause to a column as a security label column. A security label column indicates that the table is defined with multilevel security with row level granularity. The security label column contains the security label values. When you define a referential constraint, the new keywords ENFORCED or NOT ENFORCED indicate whether DB2 enforces the constraint or treats the constraint as an informational referential constraint. The default is ENFORCED. DB2 assumes that the user enforces the constraint when it is defined as NOT ENFORCED, or informational.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
CREATE TABLE (cont)	The AS IDENTITY clause has new keywords, ORDER and NO ORDER, to indicate whether the identity values must be generated in order of request. The default is NO ORDER. You can also use the new keywords NO MAXVALUE and NO MINVALUE to explicitly specify the default behavior that an identity column has no maximum or minimum value unless you specify one. In addition, you can now specify 0 as the interval between sequence values and set the minimum and maximum values to the same value, which effectively lets you define a constant sequence such that the same value is always returned. Many new clauses are also added to the ALTER TABLE statement that let you change any of the attributes of an existing identity column. When you create a table with a LOB column, you are no longer required to explicitly define a ROWID column. You can let DB2 implicitly generate one for you. An implicitly generated ROWID column is called a *hidden ROWID column*. When you create a table with a ROWID column, you are no longer required to explicitly specify GENERATED ALWAYS or GENERATED BY DEFAULT. The default is GENERATED ALWAYS.
CREATE TABLESPACE	**New clauses:** PARTITION **Deprecated clauses:** LOCKPART When defining a partitioned table space, you can specify PARTITION instead of PART to identify the partitions to create. Although the PART keyword is still supported as a synonym, PARTITION is the new preferred syntax. In Version 8, DB2 treats all partitioned table spaces as if they were defined with LOCKPART YES. You can still specify the LOCKPART clause, but it has no effect. When all the conditions for selective partition locking are met, DB2 locks only the partitions that are accessed. When the conditions for selective partition locking are not met, DB2 locks every partition of the table space.
CREATE VIEW	**New clauses:** WITH *common-table-expression* You can use the WITH clause to define a common table expression, which is like a temporary view that can be used for the duration of the CREATE VIEW statement

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
DECLARE CURSOR	**New clauses:** ASENSITIVE SCROLL SENSITIVE DYNAMIC SCROLL WITHOUT ROWSET POSITIONING WITH ROWSET POSITIONING NO SCROLL WITHOUT HOLD WITHOUT RETURN In previous releases of DB2, the sensitivity of a scrollable cursor could be SENSITIVE STATIC or INSENSITIVE. In Version 8, you can also define a scrollable cursor as ASENSITIVE or SENSITIVE DYNAMIC. For scrollable cursors, the default is ASENSITIVE. ASENSITIVE specifies the default cursor sensitivity: INSENSITIVE if the cursor is read-only and SENSITIVE DYNAMIC if it is not. SENSITIVE DYNAMIC indicates that the result table is dynamic. That is, the result table is not fixed in size when the cursor is opened and can change in size. The new WITHOUT ROWSET POSITIONING and WITH ROWSET POSITIONING clauses control whether the cursor can be used only with row-positioned or both row-positioned and rowset-positioned FETCH statements. Before Version 8, you could specify the SCROLL, WITH HOLD, and WITH RETURN clauses; however, no syntax matched the default behavior that would occur in the absence of specifying any of these clauses. In Version 8, the new clauses NO SCROLL, WITHOUT HOLD, and WITHOUT RETURN are added to denote the default behavior.
DECLARE GLOBAL TEMPORARY TABLE	**New clauses:** EXCLUDING IDENTITY COLUMN DEFAULTS EXCLUDING COLUMN DEFAULTS ON COMMIT DROP TABLE WITH NO DATA In previous releases of DB2, INCLUDING IDENTITY COLUMN DEFAULTS and INCLUDING COLUMN DEFAULTS were introduced as optional clauses. No clauses existed for the default behavior. You can now explicitly use EXCLUDING IDENTITY COLUMN DEFAULTS and EXCLUDING COLUMN DEFAULTS to specify the default behavior. The ON COMMIT DROP TABLE clause indicates that the declared global temporary table is to be dropped on a commit if no open cursors on the table are defined as WITH HOLD. The new clause WITH NO DATA is introduced to be a synonym of the existing clause DEFINITION ONLY. WITH NO DATA is the preferred syntax.
DELETE	**New clauses:** FOR ROW *n* OF ROWSET For a positioned delete in which the cursor is positioned on a rowset, you can use the new FOR ROW *n* of ROWSET clause to specify which row of the rowset is to be deleted. If the cursor is positioned on a rowset and you omit the FOR ROW *n* of ROWSET clause, all the rows of the current rowset are deleted. Thus, you can delete multiple rows with a single statement.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
DROP	**New clauses:** SEQUENCE **Changed clauses:** RESTRICT You can use the new SEQUENCE clause to drop a sequence. RESTRICT is now an optional keyword. You are no longer required to explicitly specify RESTRICT when you drop a distinct type, stored procedure, or function.
EXECUTE	**New clauses:** USING *host-variable-array* or *host-variable* USING DESCRIPTOR *descriptor-name* FOR *n* ROWS The new clauses provide support for dynamic INSERT statements that are prepared to insert multiple rows. FOR *n* ROWS specifies the number of rows that are to be inserted. The USING clause can explicitly define the host variables or host variable array that contain the values to be inserted or reference an SQLDA, which describes them.
EXECUTE IMMEDIATE	In Version 8, when you specify *host-variable*, *host-variable* must have a CLOB or a DBCLOB data type if the SQL statement that is being prepared is greater than 32-KB in length.
EXPLAIN	**New clauses:** STMTCACHE STMTID *id-host-variable* or *integer-constant* STMTCACHE STMTOKEN *token-host-variable* or *string-constant* STMTCACHE All With the new STMCACHE STMTID and STATEMENTCACHE STMTOKEN clauses, you can now explain a statement in the dynamic statement cache. You identify the cached statement to be explained by specifying its associated statement ID or statement token. With the new STMTCACHE ALL clause, you can get information about all of the SQL statements in the dynamic statement cache. The STMTCACHE ALL clause allows one row of information per SQL statement to be written to the DSN_STATEMENT_CACHE_TABLE. You can also now populate an explain table that you do not own. If DB2 finds an alias on PLAN_TABLE, DSN_STATEMMT_TABLE, DSN_FUNCTION_TABLE and the current authorization ID has sufficient SELECT and INSERT privileges, DB2 populates the table that is referenced by the alias.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
FETCH	**New clauses:** NEXT ROWSET PRIOR ROWSET FIRST ROWSET LAST ROWSET CURRENT ROWSET ROWSET STARTING AT ABSOLUTE *n* ROWSET STARTING AT ABSOLUTE *n* FOR *n* ROWS INTO *host-variable-array, ...* (or *descriptor-name*) To support rowset-positioned cursors and the retrieval of multiple rows of data from a result table with a single statement, the FETCH statement is enhanced with many new clauses. For example, you can specify FIRST ROWSET to position the cursor on the first rowset of the result table. Specifying NEXT ROWSET positions the cursor on the next rowset of the result table, relative to the current cursor position. The clause FOR *n* ROWS determines the maximum number of rows that are retrieved. The INTO clause identifies the host variable arrays that are to receive the data that is fetched. You can define the host variable arrays in your program or describe them in an SQLDA.
GET DIAGNOSTICS	**New clauses:** *statement-information* *condition-information* *combined information* The GET DIAGNOSTICS statement is enhanced to return more diagnostic information about the last SQL statement than just the number of rows that were associated with that statement. You can specify many more clauses and get information about statement items, condition items, and connection items. You no longer need to use GET DIAGNOSTICS from within an SQL procedure.
INSERT	**New clauses:** WITH *common-table-expression* VALUES(*expression, host-variable-array, ...*,) FOR *n* ROWS ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION You can use the WITH clause to define a common table expression, which is like a temporary view that can be used for the duration of the INSERT statement The new FOR *n* ROWS clause lets you insert multiple rows into a table or view. In previous versions of DB2, you could insert only one row with a single INSERT statement. The VALUES clause specifies the data that is to be inserted. The values can be specified in expressions, in a host variable array, as null, or as the default value for the column. When you insert multiple rows, the ATOMIC and NOT ATOMIC CONTINUE ON SQLEXCEPTION keywords control whether all the rows are inserted as an atomic operation. ATOMIC specifies that if the insert for any row fails, all changes made by any of the inserts, even successful ones, are undone. NOT ATOMIC CONTINUE ON SQLEXCEPTION specifies that if the insert of one row fails, the changes made for the successful inserts of other rows are not undone. The default is ATOMIC.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
LOCK TABLE	**New clauses:** PARTITION When identifying the partition of a partitioned table space to lock, you can specify PARTITION instead of PART to identify the partitions to create. Although the PART keyword is still supported as a synonym, PARTITION is the new preferred syntax.
PREPARE	**New clauses:** ASENSITIVE NO SCROLL WITHOUT ROWSET POSITIONING WITH ROWSET POSITIONING FOR SINGLE ROW FOR MULTIPLE ROWS ATOMIC NOT ATOMIC CONTINUE ON SQLEXCEPTION WITHOUT HOLD WITHOUT RETURN **Changed clauses:** SENSITIVE In previous releases of DB2, the sensitivity of a scrollable cursor could be SENSITIVE STATIC or INSENSITIVE. In Version 8, in support of dynamic scrollable cursors, you can also define a cursor as ASENSITIVE or SENSITIVE DYNAMIC. The default is ASENSITIVE. ASENSITIVE specifies the default cursor sensitivity: INSENSITIVE if the cursor is read-only and SENSITIVE DYNAMIC if it is not. SENSITIVE DYNAMIC indicates that the size of the result table is not fixed when the cursor is opened and the cursor has complete visibility to changes. The new WITHOUT ROWSET POSITIONING and WITH ROWSET POSITIONING clauses control whether the cursor can be used with only row-positioned or both row-positioned and rowset-positioned FETCH statements. If the statement that is being prepared is a dynamic INSERT statement, you can specify FOR SINGLE ROW or FOR MULTIPLE ROWS clauses to indicate whether a variable number of rows is to be provided for the INSERT statement. The default is FOR SINGLE ROW. The ATOMIC and NOT ATOMIC CONTINUE ON SQLEXCEPTION keywords control whether all the rows are inserted as an atomic operation. ATOMIC specifies that if the insert for any row fails, all changes made by any of the inserts, even successful ones, are undone. NOT ATOMIC CONTINUE ON SQLEXCEPTION specifies that if the insert of one row fails, the changes made for the successful inserts of other rows are not undone. The default is ATOMIC. Before Version 8, you could specify the SCROLL, WITH HOLD, and WITH RETURN clauses; however, no syntax matched the default behavior that would occur in the absence of specifying any of these clauses. In Version 8, the new clauses NO SCROLL, WITHOUT HOLD, and WITHOUT RETURN are added to denote the default behavior. Support for the explicit specification of the WITHOUT clauses enables you to override the specification of any WITH clauses on a DECLARE CURSOR statement. # In Version 8, when you specify FROM *host-variable*, *host-variable* must have a CLOB # or a DBCLOB data type if the SQL statement that is being prepared is greater than # 32-KB in length.

Table 23. Changes to existing SQL statements (continued)

SQL statement	Description of enhancements and notes
SELECT INTO	**New clauses:** ORDER BY The ability to specify the new ORDER BY clause lets you affect which row is returned when you use the FETCH FIRST ROW clause. The FETCH FIRST ROW clause ensures that only one row is returned when the query can result in more than a single row. When both clauses are specified, the ordering is done first on the result set and then the first row is retrieved.
SIGNAL	**New clauses:** *condition-name* Before Version 8 of DB2, the name of the SIGNAL statement was SIGNAL SQLSTATE, and you could use the statement only in a trigger body. You can now also specify SIGNAL *condition-name* to have an SQL procedure return a condition with a specific SQLSTATE and message text.
UPDATE	**New clauses:** FOR ROW *n* OF ROWSET For a positioned update in which the cursor is positioned on a row set, you can use the new FOR ROW *n* OF ROWSET clause to specify which row of the row set is to be updated. If the cursor is positioned on a row set and you omit the FOR ROW *n* of ROWSET clause, all the rows of the current row set are updated. Therefore, you can update multiple rows with a single statement.

New functions

Table 24 shows the new built-in functions in Version 8, which improve the power of the SQL language.

Table 24. New functions

Function name	Description
ASCII	Returns the leftmost character of a string expression as an integer.
CHARACTER_LENGTH	Returns the length of a string expression as a 32-bit UTF-32, 16-bit UTF-16, or byte value
DECRYPT_BIT, DECRYPT_CHAR, or DECRYPT_DB	Returns the decrypted value of an encrypted argument
ENCRYPT_TDES	Returns the argument as an encrypted value
GETHINT	Returns the embedded password hint from encrypted data, if one exists
GETVARIABLE	Returns the value of a session variable
POSITION	Returns the starting position of the first occurrence of one string within another string
SUBSTRING	Returns a string (a substring) from within another string
TIMESTAMPDIFF	Returns an estimated number of intervals of a specified type, based on the difference between two timestamps.
XML2CLOB	Converts a transient XML data type into a CLOB so that applications can access the XML data
XMLAGG	Produces a forest of XML elements from a collection of XML elements

Table 24. New functions (continued)

Function name	Description
XMLCONCAT	Concatenates a variable number of arguments to generate a forest of XML elements
XMLELEMENT	Generates an XML element from a variable number of arguments. Uses XMLATTRIBUTES to specify attributes for the XML element to be generated.
XMLFOREST	Produces a forest of XML elements that all share a specific pattern from a list of columns and expressions
XMLNAMESPACES	Declares one or more XML namespaces

Other SQL language changes

In addition to the many new SQL statements and functions, Version 8 provides other enhancements to the SQL language, as shown in Table 25.

Table 25. Other changes to SQL language

Item	Description
Expressions for sequence values	NEXT VALUE FOR *sequence-name* and PREVIOUS VALUE FOR *sequence-name* are introduced as new expressions. For the specified sequence, NEXT VALUE FOR generates and returns the next value of the sequence. PREVIOUS VALUE FOR returns the most recently generated value for the sequence.
CAST specification	When casting an operand from one data type to another, you can use the CCSID clause to explicitly specify the encoding scheme or CCSID for the result. The new CCSID clause can be specified when casting an operand into one of the following data types: CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, or DBCLOB. # The length of the result can now be specified in a specific number of units, # CODEUNITS32, CODEUNITS16, or OCTETS, if the expression is not a string # that is defined as FOR BIT DATA.

Table 25. Other changes to SQL language (continued)

Item	Description
Special registers	Version 8 of DB2 introduces several new special registers. The CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION and CURRENT REFRESH AGE special registers are added to support materialized query tables. These registers control which materialized query tables are evaluated for use when automatic query rewrite using materialized query tables is considered. The new CURRENT PACKAGE PATH special register lets you specify a list of collections to search for a package. The SET CURRENT PACKAGE PATH SQL statement, which can be used to change the value of the register, is similar to the PKLIST bind option, but the SET CURRENT PACKAGE PATH statement is processed at the server. In previous releases of DB2, the only way to switch between packages was to execute the SET CURRENT PACKAGESET statement every time you needed to use a different package. With SET CURRENT PACKAGE PATH, you can execute the statement only once, to give the server a list of package collections to search. The CURRENT SCHEMA special register enables you to specify an implicit qualifier for unqualified object names in dynamic SQL statements. Unlike the CURRENT SQLID special register, CURRENT SCHEMA affects only the implicit qualifier that is used. The register has no affect on authorization checking for dynamic SQL statements, and it is not used to determine the owner of objects that are created dynamically. Four new special registers are added to facilitate the exchange of client information that is specified for a connection: • The CURRENT CLIENT_ACCTNG special register contains the value of the accounting string. • The CURRENT CLIENT_APPLNAME special register contains the value of the application name. • The CURRENT CLIENT_USERID special register contains the value of the client user ID. • The CURRENT CLIENT_WRKSTNNAME special register contains the value of the workstation name. The values of these registers can be provided through a number of application programming interfaces.
Predicates	The IS DISTINCT FROM predicate (and its alternate form IS NOT DISTINCT FROM) is new in Version 8 to provide enhanced processing for null data values. The predicate simplifies what needs to be coded to account for null values in search conditions, especially for checking whether two expressions are equivalent or are both null.

Table 25. Other changes to SQL language (continued)

Item	Description
Session variables	Similar to special registers, session variables are another way to provide information to applications. Version 8 now supports many DB2-defined session variables that store information that can be referenced by SQL statements. In addition, you can establish up to 10 more session variables in the connection and sign-on exit routines. You can use the GETVARIABLE built-in function to retrieve the values of session variables. The DB2-defined session variables are: • SYSIBM.DATA_SHARING_GROUP_NAME • SYSIBM.PACKAGE_NAME • SYSIBM.PACKAGE_SCHEMA • SYSIBM.PACKAGE_VERSION • SYSIBM.PLAN_NAME • SYSIBM.SECLABEL • SYSIBM.SYSTEM_NAME • SYSIBM.SYSTEM_ASCII_CCSID • SYSIBM.SYSTEM_EBCDIC_CCSID • SYSIBM.SYSTEM_UNICODE_CCSID • SYSIBM.VERSION The session variables that you establish in the connection and signon exit routines are defined the SESSION schema.
Built-in functions	Many of the built-in functions now support longer input arguments. The expression for the input argument of a column function no longer is required to reference a column. Hence, column functions are renamed to *aggregate functions*. # Many of the built-in functions now support using CODUUNIT32, # CODEUNIT16, or OCTETS to specify the unit that is used to express an integer.
GROUP BY clause	DB2 is enhanced to support expressions in the GROUP BY clause. Previously, you could only specify column names.
SYSTOOLS as a schema name	In previous versions of DB2, SYSTOOLS was restricted as a schema name for distinct types, user-defined functions, stored procedures, and triggers. The only schema name that began with character string SYS that you could specify for these objects was SYSADM. Now, you can also specify SYSTOOLS if you have the SYSADM or SYSCTRL privilege.
select-statement or SELECT INTO	You can now specify an INSERT statement in the FROM clause of a select-statement or a SELECT INTO statement. Specifying an INSERT statement in the FROM clause lets you retrieve the values of the rows that are inserted into a table (such as for default values of columns, values of automatically generated columns, values of columns that are changed by a BEFORE trigger, and values that are inserted through a multiple-row insert). The keyword FINAL TABLE followed by the INSERT statement in parentheses denotes the result table that is returned to the select-statement or SELECT INTO. The result table includes all the rows that were inserted. When you specify an INSERT statement in a select-statement, you can also specify INPUT SEQUENCE in the ORDER BY clause. INPUT SEQUENCE specifies that the rows in the result table are to be in the order in which they were inserted. You can use the new WITH clause at the beginning of a select-statement to create a common table expression that can be used for the duration of the statement. Common table expressions are especially useful in bill of material applications.

Table 25. Other changes to SQL language (continued)

Item	Description
SQL procedures	Version 8 extends support for statement labels to all statements within an SQL procedure. In previous versions of DB2, only a limited number of statements, such as the assignment-statement, compound-statement, LOOP statement, and WHILE statement, could have a label. Also, two restrictions that were enforced for SQL procedures in previous releases of DB2 are removed. Starting with Version 8 of DB2: • An SQL variable can have a LOB data type. • An SQL variable and an SQL parameter for a procedure can have the same name.

Appendix D. Catalog changes

This appendix provides an overview of the changes to the catalog, as it exists in new-function mode, for Version 8 of DB2 UDB for z/OS. The following topics provide additional information:

- "New catalog tables"
- "Changed catalog tables," including dropped and moved objects
- "New indexes" on page 133
- "When catalog migration changes occur" on page 133

For a complete description of the columns of the new and changed catalog tables, see *DB2 SQL Reference*. If you are migrating to Version 8 from Version 7, see "When catalog migration changes occur" on page 133 for a summary of when the catalog changes are made.

New catalog tables

Table 26 shows new catalog tables.

Table 26. New catalog tables

Catalog table name	Description
SYSIBM.IPLIST	Allows multiple IP addresses to be specified for a given location to enable the definition of a remote DB2 data sharing group. The table is created in existing table space DSNDB06.SYSDDF.
SYSIBM.SYSSEQUENCEAUTH	Records the privileges that users hold on sequences. The table is created in existing table space DSNDB06.SYSSEQ2.

In addition, SYSIBM.SYSOBDS is a new catalog table in Version 8. The table is in table space SYSALTER. The table is not described here because it is for IBM internal use only.

Changed catalog tables

Many existing catalog tables are changed in Version 8. Table 27 shows a list of the new columns and the existing columns that are revised. Revisions to columns include new column descriptions, new values for a column, changed data types, changed column lengths, or both changed data types and lengths.

Table 27. Summary of new and revised catalog table columns

Catalog table name	New column	Revised column
IPNAMES		SECURITY_OUT LINKNAME
LOCATIONS	DBALIAS	LOCATION LINKNAME PORT TPN
LULIST		LINKNAME LUNAME

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
LUMODES		LUNAME MODENAME
LUNAMES		LUNAME SYSMODENAME
MODESELECT		AUTHID PLANNAME LUNAME MODENAME
SYSAUXRELS		TBOWNER TBNAME COLNAME AUXTBOWNER AUXTBNAME
SYSCHECKDEP		TBOWNER TBNAME COLNAME
SYSCHECKS		TBOWNER CREATOR TBNAME CHECKCONDITION
SYSCHECKS2		TBOWNER TBNAME PATHSCHEMAS
SYSCOLAUTH		GRANTOR GRANTEE CREATOR TNAME COLNAME LOCATION COLLID CONTOKEN
SYSCOLDIST		TBOWNER TBNAME NAME COLVALUE TYPE COLGROUPCOLNO
SYSCOLDIST_HIST		TBOWNER TBNAME NAME COLVALUE TYPE COLGROUPCOLNO
SYSCOLDISTSTATS		TBOWNER TBNAME NAME COLVALUE TYPE COLGROUPCOLNO

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSCOLSTATS	STATS_FORMAT	HIGHKEY HIGH2KEY LOWKEY LOW2KEY TBOWNER TBNAME NAME
SYSCOLUMNS	STATS_FORMAT PARTKEY_COLSEQ PARTKEY_ORDERING ALTEREDTS CCSID HIDDEN	NAME TBNAME TBCREATOR HIGH2KEY LOW2KEY REMARKS FOREIGNKEY LABEL DEFAULTVALUE TYPESCHEMA TYPENAME
SYSCOLUMNS_HIST	STATS_FORMAT	NAME TBNAME TBCREATOR HIGH2KEY LOW2KEY
SYSCONSTDEP		BNAME BSCHEMA DTBNAME DTBCREATOR
SYSCOPY	OLDEST_VERSION LOGICAL_PART	ICTYPE STYPE
SYSDATABASE		NAME CREATOR STGROUP CREATEDBY GROUP_MEMBER
SYSDATATYPES		SCHEMA OWNER NAME CREATEDBY SOURCESCHEMA SOURCETYPE REMARKS
SYSDBAUTH		GRANTOR GRANTEE NAME
SYSDBRM		NAME PDSNAME PLNAME PLCREATOR VERSION

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSFIELDS		TBCREATOR TBNAME NAME FLDTYPE FLDPROC PARMLIST
SYSFOREIGNKEYS		CREATOR TBNAME RELNAME COLNAME
SYSINDEXES	PADDED VERSION OLDEST_VERSION CURRENT_VERSION RELCREATED AVGKEYLEN	NAME CREATOR TBNAME TBCREATOR DBNAME INDEXSPACE CREATEDBY INDEXTYPE REMARKS
SYSINDEXES_HIST	AVGKEYLEN	NAME CREATOR TBNAME TBCREATOR
SYSINDEXPART	OLDEST_VERSION CREATEDTS AVGKEYLEN	IXNAME IXCREATOR PQTY SQTY STORNAME VCATNAME LIMITKEY
SYSINDEXPART_HIST	AVGKEYLEN	IXNAME IXCREATOR PQTY SECQTYI
SYSINDEXSTATS		OWNER NAME
SYSINDEXSTATS_HIST		OWNER NAME
SYSJARCONTENTS		JARSCHEMA JAR_ID CLASS
SYSJAROBJECTS		JARSCHEMA JAR_ID OWNER PATH

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSJAVAOPTS		JARSCHEMA JAR_ID BUILDSCHEMA BUILDNAME BUILDOWNER DBRMLIB HPJCOMPILE_OPTS BIND_OPTS PROJECT_LIB
SYSKEYCOLUSE		TBCREATOR TBNAME COLNAME
SYSKEYS		IXNAME IXCREATOR COLNAME
SYSLOBSTATS		DBNAME NAME
SYSLOBSTATS_HIST		DBNAME NAME
SYSPACKAGE	REMARKS	LOCATION COLLID NAME CONTOKEN OWNER CREATOR QUALIFIER VERSION PDSNAME GROUP_MEMBER REOPTVAR PATHSCHEMAS OPTHINT
SYSPACKAUTH		GRANTOR GRANTEE LOCATION COLLID NAME
SYSPACKDEP		BNAME BQUALIFIER BTYPE DLOCATION DCOLLID DNAME DCONTOKEN DOWNER
SYSPACKLIST		PLANNAME LOCATION COLLID NAME

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSPACKSTMT		LOCATION COLLID NAME CONTOKEN VERSION STMT ISOLATION
SYSPARMS		SCHEMA OWNER NAME SPECIFICNAME PARMNAME ROWTYPE ORDINAL TYPESCHEMA TYPENAME CCSID
SYSPKSYSTEM		LOCATION COLLID NAME CONTOKEN SYSTEM CNAME
SYSPLAN	REMARKS	NAME CREATOR BOUNDBY QUALIFIER CURRENTSERVER GROUP_MEMBER REOPTVAR PATHSCHEMAS OPTHINT
SYSPLANAUTH		GRANTOR GRANTEE NAME
SYSPLANDEP		BNAME BCREATOR BTYPE DNAME
SYSPLSYSTEM		NAME SYSTEM CNAME
SYSRELS	ENFORCED CHECKEXISTINGDATA	CREATOR TBNAME RELNAME REFTBNAME REFTBCREATOR IXOWNER IXNAME
SYSRESAUTH		GRANTOR GRANTEE QUALIFIER NAME

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSROUTINEAUTH		GRANTOR GRANTEE SCHEMA SPECIFICNAME COLLID CONTOKEN
SYSROUTINES	NUM_DEP_MQTS MAX_FAILURE PARAMETER_CCSID	SCHEMA OWNER NAME CREATEDBY SPECIFICNAME LANGUAGE COLLID SOURCESCHEMA SOURCESPECIFIC EXTERNAL_NAME WLM_ENVIRONMENT RUNOPTS REMARKS JAVA_SIGNATURE CLASS JARSCHEMA JAR_ID
SYSROUTINES_OPTS	DEBUG_MODE	SCHEMA ROUTINENAME BUILDSCHEMA BUILDNAME BUILDOWNER PRECOMPILE_OPTS COMPILE_OPTS PRELINK_OPTS BIND_OPTS SOURCEDSN
SYSROUTINES_SRC		SCHEMA ROUTINENAME CREATESTMT
SYSSCHEMAAUTH		GRANTOR GRANTEE SCHEMANAME
SYSSEQUENCES	PRECISION RESTARTWITH	SCHEMA OWNER NAME SEQTYPE SEQUENCEID CREATEDBY CYCLE CACHE ORDER CREATEDTS ALTEREDTS REMARKS
SYSSEQUENCESDEP	DTYPE BSCHEMA BNAME DSCHEMA	BSEQUENCEID DCREATOR DNAME DCOLNAME

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSSTMT		NAME PLNAME PLCREATOR TEXT ISOLATION
SYSSTOGROUP	SPACEF	NAME CREATOR VCATNAME CREATEDBY
SYSSTRINGS		TRANSPROC
SYSSYNONYMS		NAME CREATOR VCATNAME TBNAME TBCREATOR CREATEDBY
SYSTABAUTH		GRANTOR GRANTEE DBNAME SCREATOR STNAME TCREATOR TTNAME GRANTEELOCATION LOCATION COLLID CONTOKEN
SYSTABCONST		TBCREATOR TBNAME CREATOR IXOWNER IXNAME
SYSTABLEPART	LOGICAL_PART LIMITKEY_INTERNAL OLDEST_VERSION CREATEDTS AVGROWLEN	TSNAME DBNAME IXNAME IXCREATOR PQTY SQTY STORNAME VCATNAME LIMITKEY CHECKRID5B
SYSTABLEPART_HIST	AVGROWLEN	TSNAME DBNAME PQTY SECQTYI

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSTABLES	NUM_DEP_MQTS VERSION PARTKEYCOLNUM SPLIT_ROWS SECURITY_LABEL	NAME CREATOR TYPE DBNAME TSNAME EDPROC VALPROC REMARKS PARENTS CHILDREN KEYCOLUMNS STATUS LABEL CHECKFLAG CREATEDBY LOCATION TBCREATOR TBNAME CHECKS CHECKRID5B ENCODING_SCHEME TABLESTATUS
SYSTABLES_HIST		NAME CREATOR DBNAME TSNAME
SYSTABLESPACE	OLDEST_VERSION CURRENT_VERSION AVGROWLEN SPACEF	NAME CREATOR DBNAME CREATEDBY
SYSTABSTATS		DBNAME TSNAME OWNER NAME
SYSTABSTATS_HIST		DBNAME TSNAME OWNER NAME
SYSTRIGGERS		NAME SCHEMA OWNER CREATEDBY TBNAME TBOWNER TEXT REMARKS TRIGNAME
SYSUSERAUTH		GRANTOR GRANTEE

Table 27. Summary of new and revised catalog table columns (continued)

Catalog table name	New column	Revised column
SYSVIEWDEP		BNAME BCREATOR BTYPE DNAME DCREATOR BSCHEMA DTYPE
SYSVIEWS	REFRESH ENABLE MAINTENANCE REFRESH_TIME ISOLATION SIGNATURE APP_ENCODING_CCSID	NAME CREATOR TEXT PATHSCHEMAS TYPE
SYSVOLUMES		SGNAME SGCREATOR VOLID
USERNAMES		AUTHID LINKNAME NEWAUTHID PASSWORD

Some catalog table columns that were used in previous versions of DB2 are no longer used in Version 8. Table 28 shows a list of columns that are no longer used.

Table 28. Summary of catalog table columns that are no longer used

Catalog table name	Columns no longer used
SYSCOLAUTH	DATEGRANTED TIMEGRANTED
SYSCOPY	ICDATE ICTIME
SYSDATABASE	TIMESTAMP
SYSDBAUTH	DATEGRANTED TIMEGRANTED
SYSDBRM	PRECOMPTIME PRECOMPDATE
SYSPLAN	BINDDATE BINDTIME
SYSPLANAUTH	DATEGRANTED TIMEGRANTED
SYSRESAUTH	DATEGRANTED TIMEGRANTED
SYSSTOGROUP	SPCDATE
SYSTABAUTH	DATEGRANTED TIMEGRANTED
SYSTABLESPACE	LOCKPART
SYSUSERAUTH	DATEGRANTED TIMEGRANTED

In addition to the changes described in Table 27 on page 123 and Table 28 on page 132, DB2 also makes these catalog table changes:

- Stores information about field procedures on columns of views in SYSIBM.SYSFIELDS.
- Drops catalog tables SYSIBM.SYSLINKS and SYSIBM.SYSPROCEDURES. Dropping these catalog tables also causes index DSNKCX01 on SYSIBM. SYSPROCEDURES to be dropped
- Moves catalog table SYSIBM.SYSDUMMY1 from table space SYSSTR to SYSEBCDC, which is a new EBCDIC catalog table space in Version 8.
- Uses the new PARAMETER_CCSID column in SYSIBM.SYSROUTINES to record the encoding scheme for string parameters for user-defined functions and stored procedures (PARAMETER CCSID clause). Prior to Version 8 of DB2, this information was recorded in a special row in SYSIBM.SYSPARMS (row in which ROWTYPE=X and ORDINAL=0).

New indexes

Table 29 shows the new indexes in Version 8.

Table 29. New indexes

Table space DSNDB06. ...	Catalog table SYSIBM. ...	Index	Key column
SYSALTER	SYSOBDS	DSNDOB01	CREATOR.NAME
		DSNDOB02	DBID.PSID
SYSDBASE	SYSCOLAUTH	DSNACX01	CREATOR.TNAME.COLNAME
	SYSFOREIGNKEYS	DSNDRH01	CREATOR.TBNAME.RELNAME
	SYSINDEXES	DSNDXX04	INDEXTYPE
	SYSRELS	DSNDLX02	CREATOR.TBNAME
	SYSTABAUTH	DSNATX04	TCREATOR.TTNAME
	SYSTABLEPART	DSNDPX03	DBNAME.TSNAME.LOGICAL_PART
	SYSTABLES	DSNDTX03	TBCREATOR.TBNAME
SYSDDF	IPLIST	DSNDUX01	LINKNAME.IPADDR
SYSSEQ2	SYSSEQUENCEAUTH	DSNWCX01	SCHEMA.NAME
		DSNWCX02	GRANTOR.SCHEMA.NAME
		DSNWCX03	GRANTEE.SCHEMA.NAME
	SYSSEQUENCEDEP	DSNSRX02	BSCHEMA.BNAME.DTYPE
SYSVIEWS	SYSVIEWDEP	DSNGGX04	BCREATOR.BNAME.BTYPE.DTYPE

In addition, two new indexes are created on SYSIBM.SYSOBDS, an IBM internal use only catalog table, which resides in table space SYSALTER.

When catalog migration changes occur

This section briefly describes when the various catalog changes occur when you migrate an existing Version 7 DB2 subsystem. Migrating a subsystem to Version 8 requires the completion of several installation jobs that move the subsystem to compatibility mode, enabling-new-function mode, and finally to new-function mode.

When the subsystem is migrated to compatibility mode, DB2 makes the following updates:

- Creates the new table spaces and most of the new catalog tables and indexes.
- Adds new columns to existing catalog tables.
- Changes the description of existing catalog table columns.
- Revises the definition of some existing indexes.

For a complete description of the DB2 catalog as it exists in Version 8 compatibility mode at the completion of phase 1, see *DB2 Diagnosis Guide and Reference*.

When the subsystem is converted from compatibility mode to new-function mode, DB2 makes the following updates:

- Creates the remaining new catalog tables and indexes.
- Changes the data type, length, or both of some existing catalog table columns.
- Adds additional values to existing catalog table columns.
- Converts the encoding scheme of the table spaces that are converted to Unicode.
- Drops catalog tables SYSIBM.SYSLINKS and SYSIBM.SYSPROCEDURES, which includes dropping index DSNKCX01 on the SYSIBM.SYSPROCEDURES table
- Moves catalog table SYSIBM.SYSDUMMY1 to catalog table space DSNDB06.SYSEBCDC.
- Revises the definition of indexes that have VARCHAR columns from PADDED to NOT PADDED.

For detailed information about when the catalog changes occur during migration, see *DB2 Installation Guide*.

Appendix E. EXPLAIN table changes

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 149.

This appendix includes the complete definitions for a DB2 PLAN_TABLE. It also provides a description of the PLAN_TABLE columns that are new and changed for Version 8 of DB2 UDB for z/OS.

Before you can use EXPLAIN, you must create a table called PLAN_TABLE to hold the results of EXPLAIN. If you have an existing PLAN_TABLE from a subsystem that ran on a previous version of DB2, you can alter it to add the new Version 8 columns. Figure 5 on page 136 shows the format of the PLAN_TABLE. The following topics provide additional information:

- "Format of the Version 8 PLAN_TABLE"
- "Descriptions of new and changed columns in PLAN_TABLE" on page 137
- "Changed columns in DSN_STATEMNT_TABLE" on page 139
- # "New statement cache table" on page 140

Format of the Version 8 PLAN_TABLE

The Version 8 PLAN_TABLE has seven new columns, giving it a total of 58 columns. The new columns are TABLE_ENCODE, TABLE_SCCSID, TABLE_MCCSID, TABLE_DCCSID, ROUTINE_ID, CTEREF, and STMTTOKEN. Additionally, many columns in the PLAN_TABLE have new data types, as shown in Figure 5 on page 136.

```
CREATE TABLE userid.PLAN_TABLE
       (QUERYNO             INTEGER        NOT NULL,
        QBLOCKNO            SMALLINT       NOT NULL,
        APPLNAME            CHAR(8)        NOT NULL,
        PROGNAME            VARCHAR(128)   NOT NULL,
        PLANNO              SMALLINT       NOT NULL,
        METHOD              SMALLINT       NOT NULL,
        CREATOR             VARCHAR(128)   NOT NULL,
        TNAME               VARCHAR(128)   NOT NULL,
        TABNO               SMALLINT       NOT NULL,
        ACCESSTYPE          CHAR(2)        NOT NULL,
        MATCHCOLS           SMALLINT       NOT NULL,
        ACCESSCREATOR       VARCHAR(128)   NOT NULL,
        ACCESSNAME          VARCHAR(128)   NOT NULL,
        INDEXONLY           CHAR(1)        NOT NULL,
        SORTN_UNIQ          CHAR(1)        NOT NULL,
        SORTN_JOIN          CHAR(1)        NOT NULL,
        SORTN_ORDERBY       CHAR(1)        NOT NULL,
        SORTN_GROUPBY       CHAR(1)        NOT NULL,
        SORTC_UNIQ          CHAR(1)        NOT NULL,
        SORTC_JOIN          CHAR(1)        NOT NULL,
        SORTC_ORDERBY       CHAR(1)        NOT NULL,
        SORTC_GROUPBY       CHAR(1)        NOT NULL,
        TSLOCKMODE          CHAR(3)        NOT NULL,
        TIMESTAMP           CHAR(16)       NOT NULL,
        REMARKS             VARCHAR(762)   NOT NULL,
        PREFETCH            CHAR(1)        NOT NULL WITH DEFAULT,
#       COLUMN_FN_EVAL      CHAR(1)        NOT NULL WITH DEFAULT,
#       MIXOPSEQ            SMALLINT       NOT NULL WITH DEFAULT,
#       VERSION             VARCHAR(64)    NOT NULL WITH DEFAULT,
#       COLLID              VARCHAR(128)   NOT NULL WITH DEFAULT,
        ACCESS_DEGREE       SMALLINT       ,
        ACCESS_PGROUP_ID    SMALLINT       ,
        JOIN_DEGREE         SMALLINT       ,
        JOIN_PGROUP_ID      SMALLINT       ,
        SORTC_PGROUP_ID     SMALLINT       ,
        SORTN_PGROUP_ID     SMALLINT       ,
        PARALLELISM_MODE    CHAR(1)        ,
        MERGE_JOIN_COLS     SMALLINT       ,
        CORRELATION_NAME    VARCHAR(128)   ,
        PAGE_RANGE          CHAR(1)        NOT NULL WITH DEFAULT,
        JOIN_TYPE           CHAR(1)        NOT NULL WITH DEFAULT,
        GROUP_MEMBER        CHAR(8)        NOT NULL WITH DEFAULT,
        IBM_SERVICE_DATA    VARCHAR(254)   FOR BIT DATA NOT NULL WITH DEFAULT,
        WHEN_OPTIMIZE       CHAR(1)        NOT NULL WITH DEFAULT,
        QBLOCK_TYPE         CHAR(6)        NOT NULL WITH DEFAULT,
        BIND_TIME           TIMESTAMP      NOT NULL WITH DEFAULT,
        OPTHINT             VARCHAR(128)   NOT NULL WITH DEFAULT,
        HINT_USED           VARCHAR(128)   NOT NULL WITH DEFAULT,
        PRIMARY_ACCESSTYPE  CHAR(1)        NOT NULL WITH DEFAULT,
        PARENT_QBLOCKNO     SMALLINT       NOT NULL WITH DEFAULT,
        TABLE_TYPE          CHAR(1)        ,
        TABLE_ENCODE        CHAR(1)        NOT NULL WITH DEFAULT,
        TABLE_SCCSID        SMALLINT       NOT NULL WITH DEFAULT,
        TABLE_MCCSID        SMALLINT       NOT NULL WITH DEFAULT,
        TABLE_DCCSID        SMALLINT       NOT NULL WITH DEFAULT,
        ROUTINE_ID          INTEGER        NOT NULL WITH DEFAULT,
        CTEREF              SMALLINT       NOT NULL WITH DEFAULT,
        STMTTOKEN           VARCHAR(240))
     IN database-name.table-space-name
     CCSID EBCDIC;
```

Figure 5. 58-column format of PLAN_TABLE

Descriptions of new and changed columns in PLAN_TABLE

Table 30 shows the content of each of the new or changed columns for Version 8.

Table 30. Descriptions of new and changed columns in PLAN_TABLE

Column Name	Description	New or changed
QBLOCKNO	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.	Changed
TNAME	The names of the table, materialized query table, created or declared temporary table, materialized view, or materialized table expression. The value is blank if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB(qblockno). DSNWFQB(qblockno) is used to repesent the intermediate result of a UNION ALL or an outer join that is materialized. If a view is merged, the name of the view does not appear.	Changed
ACCESSTYPE	Indicates the method of accessing the new table. The possible values are: **I** Access by an index (identified in ACCESSCREATOR and ACCESSNAME) **I1** Access by a one-fetch index scan **M** Access by a multiple index scan (followed by MX, MI, or MU) **MI** Access by an intersection of multiple indexes **MU** Access by a union of multiple indexes **MX** Access by an index scan on the index that is named in ACCESSNAME **N** Access by an index scan when the matching predicate contains the IN keyword **R** Access by a table space scan **RW** Access by a work file scan of the result of a materialized user-defined table function **T** Access by a sparse index (star join work files) **V** Access by buffers for an INSERT statement within a SELECT **blank** Not applicable to the current row	Changed
REMARKS	A field into which you can insert any character string of 762 or fewer characters.	Changed
WHEN_OPTIMIZE	When the access path was determined: **blank** At bind time, using a default filter factor for any host variables, parameter markers, or special registers. **B** At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is reoptimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for reoptimization to occur. **R** At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for this to occur.	Changed

Table 30. Descriptions of new and changed columns in PLAN_TABLE (continued)

Column Name	Description	New or changed
QBLOCK_TYPE	Indicates the type of SQL operation performed for each query block. For the outermost query, this column identifies the statement type. The possible values are: **SELECT** SELECT statement **INSERT** INSERT statement **UPDATE** UPDATE statement **DELETE** DELETE statement **SELUPD** SELECT statement with FOR UPDATE OF clause **DELCUR** DELETE statement WHERE CURRENT OF CURSOR **UPDCUR** UPDATE statement WHERE CURRENT OF CURSOR **CORSUB** Correlated subselect or fullselect **NCOSUB** Noncorrelated subselect or fullselect **TABLEX** Table expression **TRIGGR** WHEN clause on CREATE TRIGGER **UNION** UNION **UNIONA** UNIONALL	Changed
TABLE_TYPE	Indicates the type of new table. The possible values are: **B** Buffers for an INSERT statement within a SELECT. **C** Common table expression. **F** Table function. **M** Materialized query table. **Q** Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materialization was virtual and not actual. Materialization can be virtual when the definition of the view or nested table expression contains a UNION ALL that is not distributed. **R** Recursive common table expression. **T** Table. **W** Work file. The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort.	Changed
TABLE_ENCODE	Indicates the encoding scheme of the table. If the table has a single CCSID, the possible values are: **A** ASCII **E** EBCDIC **U** Unicode M is the value of the column when the table contains multiple CCSID sets.	New
TABLE_SCCSID	The SBCS value of the table. If column TABLE_ENCODE is M, the value is 0.	New

Table 30. Descriptions of new and changed columns in PLAN_TABLE (continued)

Column Name	Description	New or changed
TABLE_MCCSID	The mixed value of the table. If column TABLE_ENCODE is M, the value is 0.	New
TABLE_DCCSID	The DBCS value of the table. If column TABLE_ENCODE is M, the value is 0.	New
ROUTINE_ID	The values for this column are for IBM use only.	New
CTEREF	If the referenced table is a common table expression, the value is the top-level query block number.	New
STMTTOKEN	User-specified statement token.	New

Your PLAN_TABLE can use many other formats with fewer columns. However, you should use the 58-column format because it gives you the most information. To alter an existing plan table with fewer than 58 columns to the 58-column format, follow these steps:

1. Determine whether your PLAN_TABLE has the following columns:
   - PROGNAME
   - CREATOR
   - TNAME
   - ACCESSTYPE
   - ACCESSNAME
   - REMARKS
   - COLLID
   - CORRELATION_NAME
   - IBM_SERVICE_DATA
   - OPTHINT
   - HINT_USED
2. For any columns that exist, use the values in Figure 5 on page 136 to change the data types of these columns to the appropriate Version 8 data types.
3. For any columns that are not in PLAN_TABLE, add these columns to the table, using the column definitions in Figure 5 on page 136.

Changed columns in DSN_STATEMNT_TABLE

In Version 8, DB2 UDB for z/OS introduces three changes to the statement table. The column PROGNAME has data type VARCHAR(128) instead of data type CHAR(8). The column COLLID has data type VARCHAR(128) instead of CHAR(18). The column STMT_ENCODE is a new column with data type CHAR(1). STMT_ENCODE is described in Table 31.

Table 31. Descriptions of new and changed columns in DSN_STATEMNT_TABLE

Column Name	Description
STMT_ENCODE	Indicates the encoding scheme of the statement. If the statement represents a single CCSID, the possible values are: **A** ASCII **E** EBCDIC **U** Unicode M is the value when the statement has multiple CCSID sets.

Your statement table can use the older format in which the STMT_ENCODE column does not exist, PROGNAME has a data type of CHAR(8), and COLLID has

a data type of CHAR(18). However, use the most current format because it gives you the most information. You can alter a statement table in the older format to a statement table in the current format.

New statement cache table

In Version 8, DB2 UDB for z/OS introduces a new statement cache table, DSN_STATEMENT_CACHE_TABLE, which is populated when the EXPLAIN STMTCACHE ALL statement is issued. DSN_STATEMENT_CACHE_TABLE stores information about statements in the cache.

For information about creating and using DSN_STATEMENT_CACHE_TABLE, see *DB2 SQL Reference* and *DB2 Application Programming and SQL Guide*.

Appendix F. New and changed IFCIDs

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 149.

Version 8 of DB2 contains a number of trace enhancements, including:
- Additional package accounting information
- Information about sorts
- Statistics fields for high-water marks for thread allocations
- Accumulated accounting data for DDF and RRSAF threads, aggregated by any combination of end user user ID, end user transaction name, or end user workstation name. The combination is selected through the subsystem parameter ACCUMUID.

This appendix briefly describes the new IFCIDs and the changes to the existing IFCIDs for each new function. The following topics provide additional information:
- New IFCIDs are described in Table 32.
- Changes to existing IFCIDs are described in Table 33 on page 142.

For a detailed description of the fields in each IFCID record, refer to the mapping macros data set library *prefix*.SDSNMACS.

New IFCIDs

Table 32 lists the new IFCIDs.

Table 32. New IFCIDs

IFCID	Trace	Class	Mapping macro	Description
DRDA data stream encryption				
0184	GLOBAL	9	DSNDQW02	Records information about encrypted data in data communication buffers. The trace data is in decrypted form.
Monitoring of system checkpoints and log offloads				
0335	STATISTICS	3	DSNDQW04	Records information about system checkpoints or stalled log offloads.
Improved monitoring of locking				
0337	PERFORMANCE	6	DSNDQW04	Records information about lock escalation occurrences. A record is written whenever lock escalation occurs.
	STATISTICS	3		
Work file database and TEMP database space usage				
0342	None	None	DSNDQW04	Records work file database or TEMP database space usage by agent.
Full SQL statement tracing				
0350	PERFORMANCE	3	DSNDQW04	Records the complete text of an SQL statement.

Changed IFCIDs

Version 8 of DB2 introduces Unicode support and long name support, which affect many of the trace records. In addition, Version 8 of DB2 introduces a number of changes to selected trace records.

Unicode support: You can direct DB2 to generate selected trace fields in Unicode (UTF-8). You do this by specifying YES in the UNICODE IFCIDS field of installation panel DSNTIPN. The fields that can appear in Unicode are marked with %U in the trace mapping macros and in the IFCID flat file.

Long name support: In Version 8, DB2 supports longer names for many of the DB2 objects. Because those names also appear in trace records, longer names appear in the trace macros and IFCID flat file. Fields with increased lengths are a subset of the fields that are marked with %U. Each field that can be longer has three corresponding new fields:

- A small integer field with the offset to the longer name
- A varying-length character value that consists of:
  - A small integer field with the length of the longer name
  - A character field that contains the longer name

If an original trace field is *n* bytes long, and a name is *m* bytes long, where *m>n*, the original field contains the first *n* bytes of the name, the offset field contains the offset to the full name, and the varying-length character field contains *m*, followed by the full name.

If a name fits in the original trace field, the original field contains the name, and the offset field contains 0.

Changes for READS requests: Monitor trace class 1 no longer needs to be active before you can make IFI READS requests, except for IFCID 0185.

Changes to mappings for repeating groups that contain varying-length fields: A repeating group that contains varying-length fields has a length of zero and a non-zero offset in the self-defining section. The length of each item in a repeating group is in the first two bytes of that item. See "Reading the self-defining section for variable-length data items" in Appendix D (Volume 2) of *DB2 Administration Guide* for information about mapping repeating groups of variable-length items.

Changes to selected trace records: Table 33 gives an overview of changes to specific IFCIDs. Changes to IFCID 0106, the system parameters record, are not included.

Table 33. Changed IFCIDs

IFCID	Description of changes
CARDINALITY option for a user-defined table function	
0022	For the access type field, new values for table function access and table function prefetch into a work file.
Coupling facility batching	
0002, 0003	New fields for the number of batched coupling facility write and castout requests.
Distribution statistics on non-indexed columns	
0023, 0024, 0025	New field values for RUNSTATS subtasks for distribution statistics collection.
Dynamic scrollable cursors	

Table 33. Changed IFCIDs (continued)

IFCID	Description of changes
0059, 0065	New fields for fetch sensitivity, fetch orientation, cursor scrollability, cursor sensitivity, and cursor result table type.
Greater than 32-KB SQL statements	
0063, 0140, 0141, 0142, 0145, 0168, 0316, 0317	New 4-byte length field to trace records that contain SQL statements, to support SQL statements that are greater than 32 KB.
Improved LPL recovery	
0021, 0044, 0150, 0172, 0196	New field values for the LPL recovery lock type.
SELECT from INSERT statement	
0022	For the access type field, a new value for buffers for SELECT from INSERT. For the table type field, a new value for buffers.
Materialized query tables	
0022	For the table type field, a new value for a materialized query table.
0140	For the type of privilege being checked, a new value for REFRESH TABLE.
Sequences	
0002, 0003, 0148	New fields for the number of CREATE SEQUENCE, ALTER SEQUENCE, and DROP SEQUENCE statements.
0062	For the statement type field, new values for CREATE SEQUENCE, ALTER SEQUENCE, and DROP SEQUENCE. For the object type field, a new value for a sequence.
0140	For the object type field, a new value for a sequence.
Multilevel security	
0142	New field that contains the SECLABEL.
Server support for common clients	
0169	Added field to trace translation from a location name to a DBALIAS name for outbound requests, and to trace translation from a location alias name to a location name for inbound requests.
64-bit virtual support	
0002, 0003, 0148, 0198, 0199, 0201, 0202	Counters that are related to hiperpools are removed.
0217, 0225	New fields to describe storage use above the 2-GB bar.
System-level point-in-time recovery	
0023, 0024, 0025	New field values for BACKUP SYSTEM and RESTORE SYSTEM.
Miscellaneous changes	
0001	New high-water mark statistics for the number of TSO foreground threads, batch threads, and concurrent allied threads.
0001, 0003, 0172, 0239	Additional fields for package-level accounting. Plan-level accounting and package-level accounting are separated. Package-level accounting is removed from IFCID 0003.
# 0002	Add information about false lock contention.

Table 33. Changed IFCIDs (continued)

IFCID	Description of changes
0003	For a database access thread that runs on a DB2 subsystem that is configured with DDF inactive thread support, an IFCID 0003 record is written and a new enclave is created, even if the thread must remain active, if the following conditions are true: • The associated package is bound with KEEPDYNAMIC(YES). • There are no held cursors. • There are no active declared temporary tables. • Only KEEPDYNAMIC(YES) keeps the thread from becoming inactive.
0003, 0147, 0148, 0239	Support is added for accumulated accounting data for DDF and RRSAF threads. The data is accumulated by any combination of end user user ID, end user transaction name, or end user workstation name.
# 0007	Add fields to record pages that were read through an I/O operation.
0024	New field values for sort tasks for CHECK LOB.
0028	New field values and field values to record sorts for multiple DISTINCT keywords in SQL statements.
0053, 0058, 0059, 0060, 0061, 0064, 0065, 0066, 0273, 0311	Add fields for the DRDA query command ID (CMDSRCID) and query instance ID (QRYINSID). These fields are used for enhanced internal processing of distributed SQL statements.
# 0065, 0066	Add fields to trace the implicit close of a cursor when SQLCODE +100 occurs.
# 0124, 0317	Add fields to trace prepare attributes.
# 0185 #	Add a field that indicates whether a changed data capture operation is the result of a triggered SQL statement.

Appendix G. How to use the DB2 library

Titles of books in the library begin with DB2 Universal Database for z/OS Version 8. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information. For a complete list of books in the library, and the sections in each book, see the bibliography at the back of this book.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks that are associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the following tasks for new releases of DB2 can be found in *DB2 Release Planning Guide*.

Installation: If you are involved with DB2 only to install the system, *DB2 Installation Guide* might be all you need.

If you will be using data sharing capabilities you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

If you want to set up a DB2 subsystem to meet the requirements of the Common Criteria, you need *DB2 Common Criteria Guide*, which contains information that supersedes other information in the DB2 UDB for z/OS library regarding Common Criteria.

End use: End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *DB2 Application Programming and SQL Guide*, and *DB2 SQL Reference*.

End users can also issue SQL statements through the DB2 Query Management Facility (QMF) or some other program, and the library for that licensed program might provide all the instruction or reference material they need. For a list of the titles in the DB2 QMF library, see the bibliography at the end of this book.

Application programming: Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need the same resources that end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in Java, C, or COBOL, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, say DB2 and IMS or DB2 and CICS
- How to write distributed applications across operating systemss
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers

- How to write applications in the Java programming language to access DB2 servers

The material needed for writing a host program containing SQL is in *DB2 Application Programming and SQL Guide* and in *DB2 Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 ODBC or ODBC to access DB2 servers is in *DB2 ODBC Guide and Reference*. For handling errors, see *DB2 Codes*.

If you will be working in a distributed environment, you will need *DB2 Reference for Remote DRDA Requesters and Servers*.

Information about writing applications across operating systems can be found in *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*.

System and database administration: *Administration* covers almost everything else. *DB2 Administration Guide* divides those tasks among the following sections:

- Part 2 (Volume 1) of *DB2 Administration Guide* discusses the decisions that must be made when designing a database and tells how to implement the design by creating and altering DB2 objects, loading data, and adjusting to changes.
- Part 3 (Volume 1) of *DB2 Administration Guide* describes ways of controlling access to the DB2 system and to data within DB2, to audit aspects of DB2 usage, and to answer other security and auditing concerns.
- Part 4 (Volume 1) of *DB2 Administration Guide* describes the steps in normal day-to-day operation and discusses the steps one should take to prepare for recovery in the event of some failure.
- Part 5 (Volume 2) of *DB2 Administration Guide* explains how to monitor the performance of the DB2 system and its parts. It also lists things that can be done to make some parts run faster.

If you will be using the RACF access control module for DB2 authorization checking, you will need *DB2 RACF Access Control Module Guide*.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *DB2 Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, you also need:

- *DB2 SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *DB2 Utility Guide and Reference*, which explains how to run utilities
- *DB2 Command Reference*, which explains how to run commands

If you will be using data sharing, you need *DB2 Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *DB2 Messages* and *DB2 Codes*, which list messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis: Diagnosticians detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *DB2 Diagnosis Guide and Reference*, *DB2 Messages*, and *DB2 Codes*.

Appendix H. How to obtain DB2 information

This section provides information that you can use to find valuable information about the DB2 product:

DB2 on the Web

Stay current with the latest information about DB2. View the DB2 home page on the Web. News items keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan your database management strategy.

You can view and search DB2 publications on the Web, or you can download and print many of the most current DB2 books. Follow links to other Web sites with more information about DB2 family and z/OS solutions. Access DB2 on the Web at the following Web site: www.ibm.com/software/db2zos.

DB2 publications

The publications for DB2 UDB for z/OS are available in various formats and delivery methods. IBM provides mid-version updates in softcopy on the Web and on CD-ROM.

DB2 Information Center for z/OS solutions

DB2 UDB for z/OS product information is viewable in the DB2 Information Center for z/OS solutions. The information center, introduced in Version 8 of DB2 UDB for z/OS, is a delivery vehicle for information about DB2 UDB for z/OS, IMS, QMF, and related tools. This information center enables users to search across related product information in multiple languages for data management solutions for the z/OS environment. Product technical information is provided in a format that offers more options and tools for accessing, integrating, and customizing information resources. The information center is based on Eclipse open source technology.

The DB2 Information Center for z/OS solutions is viewable at the following Web site: http://publib.boulder.ibm.com/infocenter/db2zhelp.

CD-ROMs and DVD

Books for Version 8 of DB2 UDB for z/OS are available on a CD-ROM that is included with your product shipment:

- DB2 UDB for z/OS Version 8 Licensed Library Collection, LK3T-7128, in English

The CD-ROM contains the collection of books for DB2 UDB for z/OS in PDF and BookManager formats. Periodically, IBM refreshes the books on subsequent editions of this CD-ROM.

The books for Version 8 of DB2 UDB for z/OS are also available on the following CD-ROM and DVD collection kits, which contain online books for many IBM products:

- IBM eServer zSeries Online Library: z/OS Software Products Collection, SK3T-4270, in English
- IBM eServer zSeries Online Library: z/OS Software Products DVD Collection, SK3T–4271, in English

PDF format

Many of the DB2 books are available in PDF (Portable Document Format) for viewing or printing from CD-ROM or the Web. Download the PDF books to your intranet for distribution throughout your enterprise.

BookManager format

You can use online books on CD-ROM to read, search across books, print portions of the text, and make notes in these BookManager books. Using the IBM Softcopy Reader, appropriate IBM Library Readers, or the BookManager Read product, you can view these books in the z/OS, Windows, and VM environments. You can also view and search many of the DB2 BookManager books on the Web.

DB2 education

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. IBM schedules classes are in cities all over the world. You can find class information, by country, at the IBM Learning Services Web site: www.ibm.com/services/learning.

IBM also offers classes at your location, at a time that suits your needs. IBM can customize courses to meet your exact requirements. For more information, including the current local schedule, please contact your IBM representative.

How to order the DB2 library

You can order DB2 publications and CD-ROMs through your IBM representative or the IBM branch office that serves your locality. If your location is within the United States or Canada, you can place your order by calling one of the toll-free numbers:

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-565-1234.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS option. Be prepared to give your customer number, the product number, and either the feature codes or order numbers that you want.

You can also order books from the IBM Publication Center on the Web: www.elink.ibmlink.ibm.com.

From the IBM Publication Center, you can go to the Publication Notification System (PNS). PNS users receive electronic notifications of updated publications in their profiles. You have the option of ordering the updates by using the publications direct ordering application or any other IBM publication ordering channel. The PNS application does not send automatic shipments of publications. You will receive updated publications and a bill for them if you respond to the electronic notification.

Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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 Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

This book is intended to help you plan for Version 8 of DB2 UDB for z/OS. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by DB2 Universal Database for z/OS (DB2 UDB for z/OS).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 UDB for z/OS.

However, this book also documents 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 might 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.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AD/Cycle
C/370
CICS
DB2
DB2 Connect
DB2 Universal Database
DFSMSdss
DFSMShsm
DFSORT
Distributed Relational Database Architecture
DRDA
Enterprise Storage Server
ES/3090
eServer
FlashCopy
IBM
IBMLink
IMS
iSeries
Language Environment
MVS
MVS/ESA
OS/390
Parallel Sysplex
PR/SM
RACF
System/390
VTAM
z/OS
zSeries

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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.

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library.

A

automatic query rewrite. A process that examines an SQL statement that refers to one or more base tables, and, if appropriate, rewrites the query so that it performs better. This process can also determine whether to rewrite a query so that it refers to one or more materialized query tables that are derived from the source tables.

C

clustering index. An index that determines how rows are physically ordered (*clustered*) in a table space. If a clustering index on a partitioned table is not a partitioning index, the rows are ordered in cluster sequence within each data partition instead of spanning partitions. Prior to Version 8 of DB2 UDB for z/OS, the partitioning index was required to be the clustering index.

copy pool. A named set of SMS storage groups that contains data that is to be copied collectively. A copy pool is an SMS construct that lets you define which storage groups are to be copied by using FlashCopy functions. HSM determines which volumes belong to a copy pool.

copy target. A named set of SMS storage groups that are to be used as containers for copy pool volume copies. A copy target is an SMS construct that lets you define which storage groups are to be used as containers for volumes that are copied by using FlashCopy functions.

copy version. A point-in-time FlashCopy copy that is managed by HSM. Each copy pool has a version parameter that specifies how many copy versions are maintained on disk.

D

DAD. See *Document access definition.*

data-partitioned secondary index (DPSI). A secondary index that is partitioned. The index is partitioned according to the underlying data.

data space. In releases prior to DB2 UDB for z/OS, Version 8, a range of up to 2 GB of contiguous virtual storage addresses that a program can directly manipulate. Unlike an address space, a data space can hold only data; it does not contain common areas, system data, or programs.

document access definition (DAD). Used to define the indexing scheme for an XML column or the mapping scheme of an XML collection. It can be used to enable an XML Extender column of an XML collection, which is XML formatted.

DPSI. Data-partitioned secondary index.

dynamic cursor. A named control structure that an application program uses to change the size of the result table and the order of its rows after the cursor is opened. Contrast with *static cursor.*

dynamic statement cache pool. A cache, located above the 2-GB storage line, that holds dynamic statements.

E

EB. See *exabyte.*

exabyte. For processor, real and virtual storage capacities and channel volume:
1 152 921 504 606 846 976 bytes or 2^{60}.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that is a subset of Standardized General Markup Language (SGML). The less complex nature of XML makes it easier to write applications that handle document types, to author and manage structured information, and to transmit and share structured information across diverse computing environments.

F

FlashCopy. A function on the IBM Enterprise Storage Server® that can create a point-in-time copy of data while an application is running.

forest. An ordered set of subtrees of XML nodes.

fully escaped mapping. A mapping from an SQL identifier to an XML name when the SQL identifier is a column name.

H

hiperspace. In releases prior to DB2 UDB for z/OS, Version 8, a range of up to 2 GB of contiguous virtual storage addresses that a program can use as a buffer. Like a data space, a hiperspace can hold user data; it does not contain common areas or system data. Unlike

an address space or a data space, data in a hiperspace is not directly addressable. To manipulate data in a hiperspace, users must bring the data into the address space in 4-KB blocks.

host variable array. An array of elements, each of which corresponds to a value for a column. The dimension of the array determines the maximum number of rows for which the array can be used.

I

index-controlled partitioning. A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are specified on the CREATE INDEX statement. Partition limits are saved in the LIMITKEY column of the SYSIBM.SYSINDEXPART catalog table.

insensitive cursor. A cursor that is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table has been materialized.

L

location alias. Another name by which a database server identifies itself in the network. Applications can use this name to access a DB2 database server.

M

materialized query table. A table that is used to contain information that is derived and can be summarized from one or more source tables.

N

nonpartitioned index. An index that is not physically partitioned. Both partitioning indexes and secondary indexes can be nonpartitioned.

nonpartitioned secondary index (NPSI). An index on a partitioned table space that is not the partitioning index and is not partitioned.

nonpartitioning index. See *secondary index.*

NPSI. See *nonpartitioned secondary index.*

P

partitioned index. An index that is physically partitioned. Both partitioning indexes and secondary indexes can be partitioned.

partitioning index. An index in which the leftmost columns are the partitioning columns of the table. The index can be partitioned or nonpartitioned.

R

rowset. A set of rows for which a cursor position is established.

rowset cursor. A cursor that is defined so that one or more rows can be returned as a rowset for a single FETCH statement, and the cursor is positioned on the set of rows that is fetched.

rowset-positioned access. The ability to retrieve multiple rows from a single FETCH statement.

row-positioned access. The ability to retrieve a single row from a single FETCH statement.

S

schema. (1) The organization or structure of a database. (2) A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:

```
CREATE DISTINCT TYPE C.T ...
```

secondary index. A nonpartitioning index on a partitioned table.

sensitive cursor. A cursor that is sensitive to changes that are made to the database after the result table has been materialized.

sequence. A user-defined object that generates a sequence of numeric values according to user specifications.

source table. A table that can be a base table, a view, a table expression, or a user-defined table function.

static cursor. A named control structure that does not change the size of the result table or the order of its rows after an application opens the cursor. Contrast with *dynamic cursor.*

T

table-controlled partitioning. A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are defined in the CREATE TABLE statement. Partition limits are saved in the LIMITKEY_INTERNAL column of the SYSIBM.SYSTABLEPART catalog table.

transient XML data type. A data type for XML values that exists only during query processing.

tree structure. A data structure that represents entities in nodes, with a most one parent node for each node, and with only one root node.

V

volatile table. A table for which SQL operations choose index access whenever possible.

X

XML. See *Extensible Markup Language*.

XML attribute. A name-value pair within a tagged XML element that modifies certain features of the element.

XML element. A logical structure in an XML document that is delimited by a start and an end tag. Anything between the start tag and the end tag is the content of the element.

XML node. The smallest unit of valid, complete structure in a document. For example, a node can represent an element, an attribute, or a text string.

XML publishing functions. Functions that return XML values from SQL values.

Bibliography

DB2 Universal Database for z/OS Version 8 product information:

- *DB2 Administration Guide*, SC18-7413
- *DB2 Application Programming and SQL Guide*, SC18-7415
- *DB2 Application Programming Guide and Reference for Java*, SC18-7414
- *DB2 Codes*, GC18-9603
- *DB2 Command Reference*, SC18-7416
- *DB2 Common Criteria Guide*, SC18-9672
- *DB2 Data Sharing: Planning and Administration*, SC18-7417
- *DB2 Diagnosis Guide and Reference*, LY37-3201
- *DB2 Diagnostic Quick Reference Card*, LY37-3202
- *DB2 Image, Audio, and Video Extenders Administration and Programming*, SC26-9947
- *DB2 Installation Guide*, GC18-7418
- *DB2 Licensed Program Specifications*, GC18-7420
- *DB2 Management Clients Package Program Directory*, GI10-8567
- *DB2 Messages*, GC18-9602
- *DB2 ODBC Guide and Reference*, SC18-7423
- *The Official Introduction to DB2 UDB for z/OS*
- *DB2 Program Directory*, GI10-8566
- *DB2 RACF Access Control Module Guide*, SC18-7433
- *DB2 Reference for Remote DRDA Requesters and Servers*, SC18-7424
- *DB2 Reference Summary*, SX26-3853
- *DB2 Release Planning Guide*, SC18-7425
- *DB2 SQL Reference*, SC18-7426
- *DB2 Text Extender Administration and Programming*, SC26-9948
- *DB2 Utility Guide and Reference*, SC18-7427
- *DB2 What's New?*, GC18-7428
- *DB2 XML Extender for z/OS Administration and Programming*, SC18-7431

Books and resources about related products:

APL2®

- *APL2 Programming Guide*, SH21-1072
- *APL2 Programming: Language Reference*, SH21-1061
- *APL2 Programming: Using Structured Query Language (SQL)*, SH21-1057

BookManager® READ/MVS

- *BookManager READ/MVS V1R3: Installation Planning & Customization*, SC38-2035

C language: IBM C/C++ for z/OS

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821

Character Data Representation Architecture

- *Character Data Representation Architecture Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

CICS Transaction Server for z/OS

The publication order numbers below are for Version 2 Release 2 and Version 2 Release 3 (with the release 2 number listed first).

- *CICS Transaction Server for z/OS Information Center*, SK3T-6903 or SK3T-6957.
- *CICS Transaction Server for z/OS Application Programming Guide*, SC34-5993 or SC34-6231
- *CICS Transaction Server for z/OS Application Programming Reference*, SC34-5994 or SC34-6232
- *CICS Transaction Server for z/OS CICS-RACF Security Guide*, SC34-6011 or SC34-6249
- *CICS Transaction Server for z/OS CICS Supplied Transactions*, SC34-5992 or SC34-6230
- *CICS Transaction Server for z/OS Customization Guide*, SC34-5989 or SC34-6227
- *CICS Transaction Server for z/OS Data Areas*, LY33-6100 or LY33-6103
- *CICS Transaction Server for z/OS DB2 Guide*, SC34-6014 or SC34-6252
- *CICS Transaction Server for z/OS External Interfaces Guide*, SC34-6006 or SC34-6244
- *CICS Transaction Server for z/OS Installation Guide*, GC34-5985 or GC34-6224
- *CICS Transaction Server for z/OS Intercommunication Guide*, SC34-6005 or SC34-6243
- *CICS Transaction Server for z/OS Messages and Codes*, GC34-6003 or GC34-6241
- *CICS Transaction Server for z/OS Operations and Utilities Guide*, SC34-5991 or SC34-6229

- *CICS Transaction Server for z/OS Performance Guide*, SC34-6009 or SC34-6247
- *CICS Transaction Server for z/OS Problem Determination Guide*, SC34-6002 or SC34-6239
- *CICS Transaction Server for z/OS Release Guide*, GC34-5983 or GC34-6218
- *CICS Transaction Server for z/OS Resource Definition Guide*, SC34-5990 or SC34-6228
- *CICS Transaction Server for z/OS System Definition Guide*, SC34-5988 or SC34–6226
- *CICS Transaction Server for z/OS System Programming Reference*, SC34-5595 or SC34–6233

CICS Transaction Server for OS/390
- *CICS Transaction Server for OS/390 Application Programming Guide*, SC33-1687
- *CICS Transaction Server for OS/390 DB2 Guide*, SC33-1939
- *CICS Transaction Server for OS/390 External Interfaces Guide*, SC33-1944
- *CICS Transaction Server for OS/390 Resource Definition Guide*, SC33-1684

COBOL:
- *IBM COBOL Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS Programming Guide*, SC27-1412

Database Design
- *DB2 for z/OS and OS/390 Development for Performance Volume I* by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-605-2
- *DB2 for z/OS and OS/390 Development for Performance Volume II* by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-606-0
- *Handbook of Relational Database Design* by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8

DB2 Administration Tool
- *DB2 Administration Tool for z/OS User's Guide and Reference*, available on the Web at www.ibm.com/software/data/db2imstools/library.html

DB2 Buffer Pool Analyzer for z/OS
- *DB2 Buffer Pool Tool for z/OS User's Guide and Reference*, available on the Web at www.ibm.com/software/data/db2imstools/library.html

DB2 Connect
- *IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition*, GC09-4833
- *IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition*, GC09-4834
- *IBM DB2 Connect User's Guide*, SC09-4835

DB2 DataPropagator™
- *DB2 Universal Database Replication Guide and Reference*, SC27-1121

DB2 Performance Expert for z/OS, Version 1

The following books are part of the DB2 Performance Expert library. Some of these books include information about the following tools: IBM DB2 Performance Expert for z/OS; IBM DB2 Performance Monitor for z/OS; and DB2 Buffer Pool Analyzer for z/OS.
- *OMEGAMON Buffer Pool Analyzer User's Guide*, SC18-7972
- *OMEGAMON Configuration and Customization*, SC18-7973
- *OMEGAMON Messages*, SC18-7974
- *OMEGAMON Monitoring Performance from ISPF*, SC18-7975
- *OMEGAMON Monitoring Performance from Performance Expert Client*, SC18-7976
- *OMEGAMON Program Directory*, GI10-8549
- *OMEGAMON Report Command Reference*, SC18-7977
- *OMEGAMON Report Reference*, SC18-7978
- *Using IBM Tivoli OMEGAMON XE on z/OS*, SC18-7979

DB2 Query Management Facility (QMF™) Version 8.1
- *DB2 Query Management Facility: DB2 QMF High Performance Option User's Guide for TSO/CICS*, SC18-7450
- *DB2 Query Management Facility: DB2 QMF Messages and Codes*, GC18-7447
- *DB2 Query Management Facility: DB2 QMF Reference*, SC18-7446
- *DB2 Query Management Facility: Developing DB2 QMF Applications*, SC18-7651
- *DB2 Query Management Facility: Getting Started with DB2 QMF for Windows and DB2 QMF for WebSphere*, SC18-7449
- *DB2 Query Management Facility: Getting Started with DB2 QMF Query Miner*, GC18-7451
- *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*, GC18-7444
- *DB2 Query Management Facility: Installing and Managing DB2 QMF for Windows and DB2 QMF for WebSphere*, GC18-7448

- *DB2 Query Management Facility: Introducing DB2 QMF*, GC18-7443
- *DB2 Query Management Facility: Using DB2 QMF*, SC18-7445
- *DB2 Query Management Facility: DB2 QMF Visionary Developer's Guide*, SC18-9093
- *DB2 Query Management Facility: DB2 QMF Visionary Getting Started Guide*, GC18-9092

DB2 Redbooks™

For access to all IBM Redbooks about DB2, see the IBM Redbooks Web page at www.ibm.com/redbooks

DB2 Server for VSE & VM
- *DB2 Server for VM: DBS Utility*, SC09-2983

DB2 Universal Database Cross-Platform information
- *IBM DB2 Universal Database SQL Reference for Cross-Platform Development*, available at www.ibm.com/software/data/developer/cpsqlref/

DB2 Universal Database for iSeries

The following books are available at www.ibm.com/iseries/infocenter
- *DB2 Universal Database for iSeries Performance and Query Optimization*
- *DB2 Universal Database for iSeries Database Programming*
- *DB2 Universal Database for iSeries SQL Programming Concepts*
- *DB2 Universal Database for iSeries SQL Programming with Host Languages*
- *DB2 Universal Database for iSeries SQL Reference*
- *DB2 Universal Database for iSeries Distributed Data Management*
- *DB2 Universal Database for iSeries Distributed Database Programming*

DB2 Universal Database for Linux, UNIX, and Windows:
- *DB2 Universal Database Administration Guide: Planning*, SC09-4822
- *DB2 Universal Database Administration Guide: Implementation*, SC09-4820
- *DB2 Universal Database Administration Guide: Performance*, SC09-4821
- *DB2 Universal Database Administrative API Reference*, SC09-4824
- *DB2 Universal Database Application Development Guide: Building and Running Applications*, SC09-4825
- *DB2 Universal Database Call Level Interface Guide and Reference, Volumes 1 and 2*, SC09-4849 and SC09-4850
- *DB2 Universal Database Command Reference*, SC09-4828
- *DB2 Universal Database SQL Reference Volume 1*, SC09-4844
- *DB2 Universal Database SQL Reference Volume 2*, SC09-4845

Device Support Facilities
- *Device Support Facilities User's Guide and Reference*, GC35-0033

DFSMS

These books provide information about a variety of components of DFSMS, including z/OS DFSMS, z/OS DFSMSdfp™, z/OS DFSMSdss™, z/OS DFSMShsm, and z/OS DFP.
- *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394
- *z/OS DFSMSdss Storage Administration Guide*, SC35-0423
- *z/OS DFSMSdss Storage Administration Reference*, SC35-0424
- *z/OS DFSMShsm Managing Your Own Data*, SC35-0420
- *z/OS DFSMSdfp: Using DFSMSdfp in the z/OS Environment*, SC26-7473
- *z/OS DFSMSdfp Diagnosis Reference*, GY27-7618
- *z/OS DFSMS: Implementing System-Managed Storage*, SC27-7407
- *z/OS DFSMS: Macro Instructions for Data Sets*, SC26-7408
- *z/OS DFSMS: Managing Catalogs*, SC26-7409
- *z/OS MVS: Program Management User's Guide and Reference*, SA22-7643
- *z/OS MVS Program Management: Advanced Facilities*, SA22-7644
- *z/OS DFSMSdfp Storage Administration Reference*, SC26-7402
- *z/OS DFSMS: Using Data Sets*, SC26-7410
- *DFSMSdfp Advanced Services* , SC26-7400
- *DFSMS/MVS: Utilities*, SC26-7414

DFSORT
- *DFSORT Application Programming: Guide*, SC33-4035
- *DFSORT Installation and Customization*, SC33-4034

Distributed Relational Database Architecture

- *Open Group Technical Standard*; the Open Group presently makes the following DRDA books available through its Web site at www.opengroup.org
  - *Open Group Technical Standard, DRDA Version 3 Vol. 1: Distributed Relational Database Architecture*
  - *Open Group Technical Standard, DRDA Version 3 Vol. 2: Formatted Data Object Content Architecture*
  - *Open Group Technical Standard, DRDA Version 3 Vol. 3: Distributed Data Management Architecture*

Domain Name System
- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly*, ISBN 0-59600-158-4

Education
- Information about IBM educational offerings is available on the Web at http://www.ibm.com/software/sw-training/
- A collection of glossaries of IBM terms is available on the IBM Terminology Web site at www.ibm.com/ibm/terminology/index.html

eServer™ zSeries
- *IBM eServer zSeries Processor Resource/System Manager Planning Guide*, SB10-7033

Fortran: VS Fortran
- *VS Fortran Version 2: Language and Library Reference*, SC26-4221
- *VS Fortran Version 2: Programming Guide for CMS and MVS*, SC26-4222

High Level Assembler
- *High Level Assembler for MVS and VM and VSE Language Reference*, SC26-4940
- *High Level Assembler for MVS and VM and VSE Programmer's Guide*, SC26-4941

ICSF
- *z/OS ICSF Overview*, SA22-7519
- *Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521

IMS Version 8

IMS product information is available on the IMS Library Web page, which you can find at www.ibm.com/ims
- *IMS Administration Guide: System*, SC27-1284
- *IMS Administration Guide: Transaction Manager*, SC27-1285
- *IMS Application Programming: Database Manager*, SC27-1286
- *IMS Application Programming: Design Guide*, SC27-1287
- *IMS Application Programming: Transaction Manager*, SC27-1289
- *IMS Command Reference*, SC27-1291
- *IMS Customization Guide*, SC27-1294
- *IMS Install Volume 1: Installation Verification*, GC27-1297
- *IMS Install Volume 2: System Definition and Tailoring*, GC27-1298
- *IMS Messages and Codes Volumes 1 and 2*, GC27-1301 and GC27-1302
- *IMS Open Transaction Manager Access Guide and Reference*, SC18-7829
- *IMS Utilities Reference: System*, SC27-1309

General information about IMS Batch Terminal Simulator for z/OS is available on the Web at www.ibm.com/software/data/db2imstools/library.html

IMS DataPropagator
- *IMS DataPropagator for z/OS Administrator's Guide for Log*, SC27-1216
- *IMS DataPropagator: An Introduction*, GC27-1211
- *IMS DataPropagator for z/OS Reference*, SC27-1210

ISPF
- *z/OS ISPF Dialog Developer's Guide*, SC23-4821
- *z/OS ISPF Messages and Codes*, SC34-4815
- *z/OS ISPF Planning and Customizing*, GC34-4814
- *z/OS ISPF User's Guide Volumes 1 and 2*, SC34-4822 and SC34-4823

Language Environment®
- *Debug Tool User's Guide and Reference*, SC18-7171
- *Debug Tool for z/OS and OS/390 Reference and Messages*, SC18-7172
- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562

MQSeries®
- *MQSeries Application Messaging Interface*, SC34-5604

- *MQSeries for OS/390 Concepts and Planning Guide*, GC34-5650
- *MQSeries for OS/390 System Setup Guide*, SC34-5651

National Language Support
- *National Language Design Guide Volume 1*, SE09-8001
- *IBM National Language Support Reference Manual Volume 2*, SE09-8002

NetView®
- *Tivoli NetView for z/OS Installation: Getting Started*, SC31-8872
- *Tivoli NetView for z/OS User's Guide*, GC31-8849

Microsoft® ODBC

Information about Microsoft ODBC is available at http://msdn.microsoft.com/library/

Parallel Sysplex® Library
- *System/390 9672 Parallel Transaction Server, 9672 Parallel Enterprise Server, 9674 Coupling Facility System Overview For R1/R2/R3 Based Models*, SB10-7033
- *z/OS Parallel Sysplex Application Migration*, SA22-7662
- *z/OS Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism*, SA22-7661
- *z/OS Parallel Sysplex Test Report*, SA22-7663

The *Parallel Sysplex Configuration Assistant* is available at www.ibm.com/s390/pso/psotool

PL/I: Enterprise PL/I for z/OS
- *IBM Enterprise PL/I for z/OS Language Reference*, SC27-1460
- *IBM Enterprise PL/I for z/OS Programming Guide*, SC27-1457

PL/I: PL/I for MVS & VM
- *PL/I for MVS & VM Programming Guide*, SC26-3113

SMP/E
- *SMP/E for z/OS and OS/390 Reference*, SA22-7772
- *SMP/E for z/OS and OS/390 User's Guide*, SA22-7773

Storage Management
- *z/OS DFSMS: Implementing System-Managed Storage*, SC26-7407
- *MVS/ESA Storage Management Library: Managing Data*, SC26-7397
- *MVS/ESA Storage Management Library: Managing Storage Groups*, SC35-0421
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide*, GC26-7398

System Network Architecture (SNA)
- *SNA Formats*, GA27-3136
- *SNA LU 6.2 Peer Protocols Reference*, SC31-6808
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA/Management Services Alert Implementation Guide*, GC31-6809

TCP/IP
- *IBM TCP/IP for MVS: Customization & Administration Guide*, SC31-7134
- *IBM TCP/IP for MVS: Diagnosis Guide*, LY43-0105
- *IBM TCP/IP for MVS: Messages and Codes*, SC31-7132
- *IBM TCP/IP for MVS: Planning and Migration Guide*, SC31-7189

TotalStorage™ Enterprise Storage Server
- *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy*, SG24-5680
- *Enterprise Storage Server Introduction and Planning*, GC26-7444
- *IBM RAMAC Virtual Array*, SG24-6424

Unicode
- *z/OS Support for Unicode: Using Conversion Services*, SA22-7649

Information about Unicode, the Unicode consortium, the Unicode standard, and standards conformance requirements is available at www.unicode.org

VTAM
- *Planning for NetView, NCP, and VTAM*, SC31-8063
- *VTAM for MVS/ESA Diagnosis*, LY43-0078
- *VTAM for MVS/ESA Messages and Codes*, GC31-8369
- *VTAM for MVS/ESA Network Implementation Guide*, SC31-8370
- *VTAM for MVS/ESA Operation*, SC31-8372
- *z/OS Communications Server SNA Programming*, SC31-8829
- *z/OS Communicatons Server SNA Programmer's LU 6.2 Reference*, SC31-8810
- *VTAM for MVS/ESA Resource Definition Reference*, SC31-8377

WebSphere® family

- *WebSphere MQ Integrator Broker: Administration Guide*, SC34-6171
- *WebSphere MQ Integrator Broker for z/OS: Customization and Administration Guide*, SC34-6175
- *WebSphere MQ Integrator Broker: Introduction and Planning*, GC34-5599
- *WebSphere MQ Integrator Broker: Using the Control Center*, SC34-6168

z/Architecture™

- *z/Architecture Principles of Operation*, SA22-7832

z/OS

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS C/C++ User's Guide*, SC09-4767
- *z/OS Communications Server: IP Configuration Guide*, SC31-8875
- *z/OS DCE Administration Guide*, SC24-5904
- *z/OS DCE Introduction*, GC24-5911
- *z/OS DCE Messages and Codes*, SC24-5912
- *z/OS Information Roadmap*, SA22-7500
- *z/OS Introduction and Release Guide*, GA22-7502
- *z/OS JES2 Initialization and Tuning Guide*, SA22-7532
- *z/OS JES3 Initialization and Tuning Guide*, SA22-7549
- *z/OS Language Environment Concepts Guide*, SA22-7567
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS Language Environment Debugging Guide*, GA22-7560
- *z/OS Language Environment Programming Guide*, SA22-7561
- *z/OS Language Environment Programming Reference*, SA22-7562
- *z/OS Managed System Infrastructure for Setup User's Guide*, SC33-7985
- *z/OS MVS Diagnosis: Procedures*, GA22-7587
- *z/OS MVS Diagnosis: Reference*, GA22-7588
- *z/OS MVS Diagnosis: Tools and Service Aids*, GA22-7589
- *z/OS MVS Initialization and Tuning Guide*, SA22-7591
- *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- *z/OS MVS Installation Exits*, SA22-7593
- *z/OS MVS JCL Reference*, SA22-7597
- *z/OS MVS JCL User's Guide*, SA22-7598
- *z/OS MVS Planning: Global Resource Serialization*, SA22-7600
- *z/OS MVS Planning: Operations*, SA22-7601
- *z/OS MVS Planning: Workload Management*, SA22-7602
- *z/OS MVS Programming: Assembler Services Guide*, SA22-7605
- *z/OS MVS Programming: Assembler Services Reference, Volumes 1 and 2*, SA22-7606 and SA22-7607
- *z/OS MVS Programming: Authorized Assembler Services Guide*, SA22-7608
- *z/OS MVS Programming: Authorized Assembler Services Reference Volumes 1-4*, SA22-7609, SA22-7610, SA22-7611, and SA22-7612
- *z/OS MVS Programming: Callable Services for High-Level Languages*, SA22-7613
- *z/OS MVS Programming: Extended Addressability Guide*, SA22-7614
- *z/OS MVS Programming: Sysplex Services Guide*, SA22-7617
- *z/OS MVS Programming: Sysplex Services Reference*, SA22-7618
- *z/OS MVS Programming: Workload Management Services*, SA22-7619
- *z/OS MVS Recovery and Reconfiguration Guide*, SA22-7623
- *z/OS MVS Routing and Descriptor Codes*, SA22-7624
- *z/OS MVS Setting Up a Sysplex*, SA22-7625
- *z/OS MVS System Codes* SA22-7626
- *z/OS MVS System Commands*, SA22-7627
- *z/OS MVS System Messages Volumes 1-10*, SA22-7631, SA22-7632, SA22-7633, SA22-7634, SA22-7635, SA22-7636, SA22-7637, SA22-7638, SA22-7639, and SA22-7640
- *z/OS MVS Using the Subsystem Interface*, SA22-7642
- *z/OS Planning for Multilevel Security and the Common Criteria*, SA22-7509
- *z/OS RMF User's Guide*, SC33-7990
- *z/OS Security Server Network Authentication Server Administration*, SC24-5926
- *z/OS Security Server RACF Auditor's Guide*, SA22-7684
- *z/OS Security Server RACF Command Language Reference*, SA22-7687
- *z/OS Security Server RACF Macros and Interfaces*, SA22-7682
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *z/OS Security Server RACF System Programmer's Guide*, SA22-7681
- *z/OS Security Server RACROUTE Macro Reference*, SA22-7692
- *z/OS Support for Unicode: Using Conversion Services*, SA22-7649
- *z/OS TSO/E CLISTs*, SA22-7781
- *z/OS TSO/E Command Reference*, SA22-7782

- *z/OS TSO/E Customization*, SA22-7783
- *z/OS TSO/E Messages*, SA22-7786
- *z/OS TSO/E Programming Guide*, SA22-7788
- *z/OS TSO/E Programming Services*, SA22-7789
- *z/OS TSO/E REXX Reference*, SA22-7790
- *z/OS TSO/E User's Guide*, SA22-7794
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS UNIX System Services User's Guide*, SA22-7801

Index

Numerics

A

B

C

D

E

F

G

T

U

V

W

X

Program Number: 5625-DB2

Printed in USA

SC18-7425-05